Middle-East Journal of Scientific Research 15 (9): 1264-1271, 2013
ISSN 1990-9233
© IDOSI Publications, 2013
DOI: 10.5829/idosi.mejsr.2013.15.9.11480

Efficient Doubletree: An Algorithm for Large-Scale Topology Discovery

Ranjit Kumar Nukathati

Department of Computer Science, JNTU College of Engineering, Anantapur, India

Abstract: Monitoring Internet topology was a tractable problem. Topology discovery systems are starting to be introduced in the form of easily and widely deployed software. Unfortunately, they have a problem of how to perform such measurements efficiently and in a network-friendly manner**.** When scaled up, such methods will generate so much traffic that they will begin to resemble distributed denial-of-service attacks. The existing Doubletree algorithm faces the problem of redundancy**.** To solve this, the Efficient Doubletree algorithm is proposed which reduces redundancy while maintaining nearly the same level of nodes and link coverage. Algorithm quantifies the amount of redundancy in classic Internet topology discovery approaches by taking into account the perspective of a single monitor (intramonitor) and that of an entire system (intermonitor). Efficient Doubletree simultaneously meets the conflicting demands of reducing intramonitor and intermonitor redundancy. In both the above cases a hop in the middle that is between monitor and destination will be selected by Hop Selection to increase efficiency.

Key words: Distributed denial-of-service(DDoS) attack · Efficient Doubletree · forward probing · Backward probing · Stopping rule

INTRODUCTION

Systems for active measurements in the Internet are undergoing a dramatic change. Whereas the present generation of systems operates on largely dedicated hosts, that numbering between 20 to 200, a new generation of easily downloadable measurement software means that infrastructures based on thousands of hosts could spring up overnight. Unless we carefully controlled these new systems have the potential to impose a heavy load on parts of the network that are being measured. They also have the capable to raise alarms as their traffic can easily resemble a distributed denial-of-service (DDoS) attack. This paper examines the problem and proposes and evaluates, through simulations and prototype deployment, an algorithm for Controlling one of the most common forms of active measurement: *traceroute*.

There are a number of systems active today that aim to elicit the Internet topology at the Internet protocol (IP) interface level. The most extensive tracing system is CAIDA's *skitter*. It uses 24 monitors, each monitor targeting on the order of one million destinations. In the fashion of skitter, *scamper* uses several monitors to traceroute IPv6 addresses. Some other well-known systems such as the RIPE NCC's *TTM service* and the NLANR *AMP* have larger numbers of monitors (between 100 and 200) and conduct traces in a full mesh but avoid tracing to outside destinations. *Scriptroute* is a system that allows an ordinary Internet user to perform network measurements from several distributed points. It proposes remote measurement execution on nodes of *PlanetLab*, through a daemon that implements ping, traceroute, hop-by-hop bandwidth measurement and other number of utilities.Removing spatial bias is not the only reason to employ measurement systems that use a larger number of monitors. Using with more monitors to probe the same space, each one can take a small portion and probe it more frequently. In network dynamics, that might be missed by smaller systems and can more readily be captured by the larger ones, while keeping constant as the workload per monitor. The idea of releasing easily deployable measurement software is not new. To the best of our knowledge, the idea of incorporating a traceroute monitor into a screen saver was first discussed in a paper by Cheswick in 2000. Since that time, a number of measurement tools have been released to the public in the form of screen savers or daemons, such as *Grenouille* or *NETI@home*. In the summer of 2004, the first tracerouting

Corresponding Author: Ranjit Kumar Nukathati, Department of Computer Science, JNTU College of Engineering, Anantapur, India.

tool of this type was made available: *DIMES* conducts traceroutes and pings from, at the time of writing this paper, 8700 agents distributed over five continents. Given that large-scale network mapping is emerging, contemplating such a measurement system demands attention to effi-ciency, in order to avoid generating undesirable network load.Save for our work, described here and first presented at ACM SIGMETRICS 2005, this issue has not been systematically tackled by the research community.

Our first contribution is to measure the extent to which classic topology discovery systems involve duplicated effort. By classic topology discovery, such as skitter, tracing from a small number of monitors to a large set of common destinations. We define two metrics to estimate duplicated effort in such systems: the quantity of measurements made by an individual monitor that replicate its own work (*intramonitor redunancy*) and, the quantity of measurements made by multiple monitors that replicate each other's work (*intermonitor redundancy*).

Using skitter data from August 2004, we quantify both kinds of redundancy. We show that intramonitor redundancy is high close to each monitor. This fact is not surprising given the treelike structure of routes emanating from a single monitor. Further, with respect to intermonitor redundancy, we find that most interfaces are visited by all monitors, especially when those interfaces are close to destinations. This latter form of redundancy is also potentially quite large, since it would be expected to grow linearly with the number of monitors in future large-scale measurement systems. Our analysis of the nature of redundant probing suggests more efficient algorithms for topology discovery. In particular, our second contribution is to propose and evaluate an algorithm called *Efficient Doubletree*. Efficient Doubletree takes advantage of the tree-like structure of routes, either emanating from a single source towards multiple destinations or converging from multiple sources towards a single destination, to avoid duplication of the effort. Unfortunately, the general strategies for reducing these two kinds of redundancy are in conflict. On the one hand, intramonitor redundancy is reduced by starting probing far from the monitor and working backwards along the tree-like structure that is rooted at that monitor. Once an interface is encountered and that has already been discovered by the monitor, then probing stops. On the other hand, intermonitor redundancy is reduced by probing forwards towards a destination until encountering a previously seen interface. The intermonitor redundancy reduction implies that monitors share information about what they have already

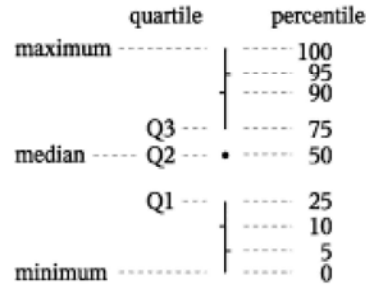

Fig. 1: Quantiles Key

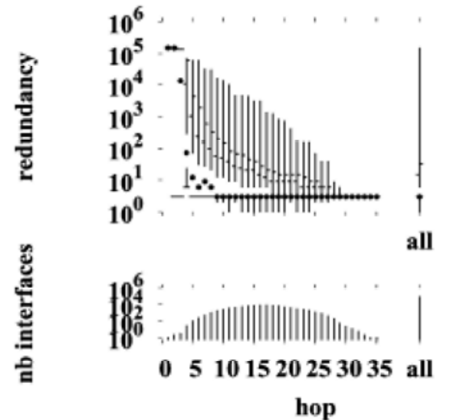

Fig. 2: Skitter intramonitor redundancy for the champagne monitor

discovered. We show, balancing these conflicting strategies in Efficient Doubletree. In Efficient Doubletree, probing starts at a distance that is intermediate between monitor and destination. We demonstrate methods for choosing this distance and we then evaluate the resulting performance of Efficient Doubletree. Despite the challenge inherent in reducing both forms of redundancy simultaneously we show in simulations that probing via Efficient Doubletree can reduce measurement load by approximately 76%, while maintaining interface and link coverage above 90%.

Metrics to Evaluate Topology Discovery Tools: This section introduces two metrics that allow one to evaluate a distributed tracerouting system. We evaluate such a system by considering the redundancy (i.e., the duplication of effort) at two levels. One is the individual level of a single monitor, considered in isolation from Fig. 2. Skitter intramonitor redundancy for the champagne monitor. This intramonitor redundancy is measured by the number of times the same monitor visits an interface. This intermonitor redundancy is measured by the number of monitors that visit an interface counting only once

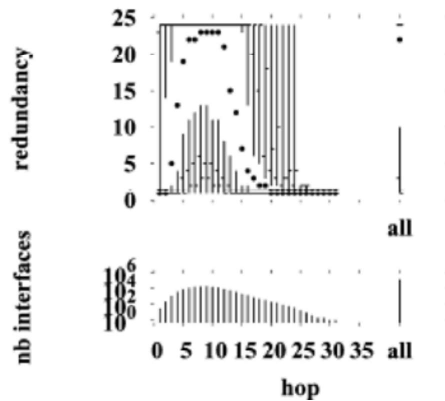

Fig. 3: Skitter intermonitor redundancy

each that has nonzero intramonitor redundancy for that interface. By separating the two levels, we separate the problem of redundancy into two problems that can be treated somewhat separately. Each monitor can act on its own to reduce its intramonitor redundancy, but cooperation between monitors is required to reduce intermonitor redundancy. Fig. 1 provides a key to reading the quantile plots found in Figs. 2 and 3 and figures found later in this paper. A dot marks the median (the 2nd quartile or 50th percentile). The vertical line below the dot delineates the range from the minimum to the 1st quartile and leaves a space from the 1st to the 2nd quartile. The space above the dot runs from the 2nd to the 3rd quartile and the line above that extends from the 3rd quartile to the maximum. Small tick bars to either side of the lines mark some additional percentiles: bars to the left for the 10th and 90th and bars to the right for the 5th and 95th.

In the figures, each quantile plot sits directly above an accompanying bar chart that indicates the quantity of data upon which the quantiles were based. And for each hop count, the bar chart displays the number of interfaces at that distance. For these bar Fig. 3. Skitter intermonitor redundancy. charts, a log scale is used on the vertical axis. This allows us to identify quantiles that are based upon very few interfaces (fewer than 20, for instance) and so for which the values risk being somewhat arbitrary. In addition, each plot has a separate bar to the right, labeled “all,” that shows the quantiles for all interfaces taken together (upper part of the plot) and the number of discovered interfaces (lower part of the plot).

Intramonitor Redundacy: Intramonitor redundancy occurs in the context of the tree-like graph that is generated when all traceroutes originate at a single point. Since there are fewer number of interfaces closer to the monitor and those interfaces will tend to be visited more frequently. In extreme cases, if there is a single gateway router between the monitor and the rest of the Internet, the single IP address of the outgoing interface belonging to that router should show up in every one of the traceroutes. We measure intramonitor redundancy by considering all traceroutes from the monitor to the shared destinations, whether there may be problems with a traceroute such as illegal addresses, or not. Having calculated the intramonitor redundancy for each interface, we organizes results by the distance of the interfaces from the monitor. We measure distance exactly by hop count. Since the same interface can appear at a number of different hop counts from a monitor, for instance, if routes change between the traceroutes, we arbitrarily attribute to each interface the hop count at which it was first visited. This process yields, for each hop count, a set of interfaces that we sort by number of visits. The we plot, hop-by-hop, the redundancy distribution for interfaces at each hop count. Fig. 2 shows the intramonitor redundancy metric applied to the champagne monitor.

Intermonitor Redundancy: Intermonitor redundancy occurs when multiple monitors visit the same interface. The degree of such redundancy is of keen interest to us when we envisage increasing the number of monitors by several orders of magnitude. We calculate the intermonitor redundancy for each interface by counting the number of monitors that have visited it. And a monitor can be counted at most once towards an interface’s intermonitor redundancy, even if it has visited that interface multiple number of times. For a given interface, the redundancy is calculated just once with respect to the entirety of the monitors: it does not vary from monitor to monitor as does intramonitor redundancy. However, what does vary depending upon the monitor is whether the particular interface is seen and at what distance. In order to attribute a single distance to an interface, a distance that does not depend upon the perspective of a single monitor but that nonetheless has meaning when examining the effects of distance on redundancy, we attribute the minimum distance at which an interface has been seen among all the monitors. Fig. 3 shows the intermonitor redundancy metric applied to the 24 skitter monitors.

Discovering Process Using Efficient Doubletree: The *Efficient Doubletree* algorithm, which is part of the Traceroute project, is the first attempt to efficiently

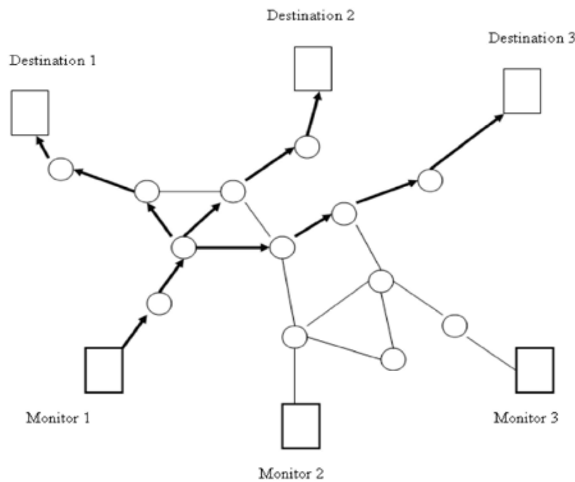

Fig. 3.1(a): Monitor-rooted tree

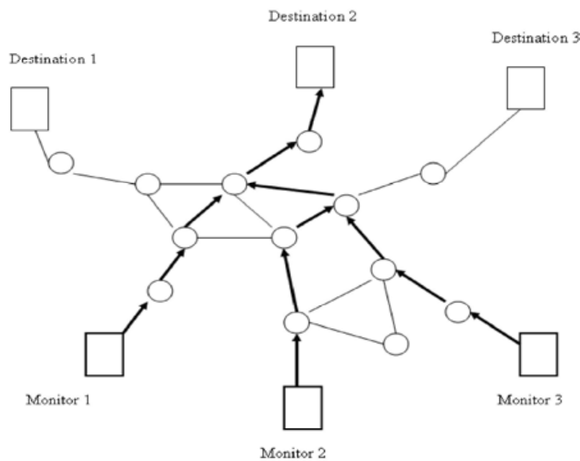

Fig. 3.1(b): Destinated-rooted tree

perform large-scale topology discovery in a network-friendly manner through co-operation between monitors. A tree can be used by a probing algorithm to keep track of its discovery progress. The rule is that probing is carried out from the leaves to the root, i.e. decreasing probe packet TTLs, as long as it is probing a previously unknown part of the tree. It then stops when a previously discovered node is encountered. The assumption is made that the remaining path to the root is known, leading to potential coverage loss. Note that, in the context of Internet topology, discovering a new node would correspond to a router's interface responding to a probe packet within the constraints. Efficient Doubletree tackles both types of redundancy (i.e. Intra-monitor redundancy and inter-monitor redundancy) and is based on the treelike structure of routes emanating from a single source or converging on a same destination. For this reason, Doubletree uses two trees, one *monitor-rooted* and one *destination-rooted* trees as illustrated by Figure 3.1(a) and 3.1(b) The monitor-rooted tree is composed of outgoing routes from a single monitor to multiple destinations. The destination-rooted tree is composed of routes from multiple monitors converging to a common destination. The stopping rule based on the former aims at reducing intra-monitor redundancy and the latter which requires inter-monitor communication in order to reduce their shared redundancy.

Two-Phase Probing: Efficient Doubletree acts in two phases:

- Forward probing proceeds from an initial hop count h to h+1, h+2 and so forth, applying a stopping rule based on the destination-rooted tree.
- Backward probing then follows by taking the hop count back to h-1, h-2, etc., applying a stopping rule based on the monitor-rooted tree.

During the backward probing phase, once the root or stopping rule is reached, the algorithm moves on to the next destination and restarts probing at initial distance h. In the special case where there is no response at a certain distance, this distance is halved and halved again until there is a reply. Probing then continues forward and backward from that point.

Stop Sets and Simple Stopping Rule: Rather than maintaining information about the tree structure, it is sufficient for the stopping rules to store sets of *(interface, root)* pairs, the *root* being the root of the tree in question, i.e. the monitor- or destination-based tree. These sets are called *stop sets*. A single monitor uses two distinct stop sets:

- The first stop set is used when probing forwards and is called the *global stop set*.
- The second stop set is used when probing backwards and is called the *local stop set*. This set can be reduced to a list of interfaces since the root item never changes as it is the monitor itself.

The stopping rule for each phase is simple: it is to stop when the encountered pair is already a member of the relevant set, otherwise it is added.

Algorithm: Efficient Doubletree: Algorithm 1 is the formal definition of Efficient Doubletree and assumes the existence of two functions:

- *Response()* returns true if an interface replies to at least one probe sent to it.
- *Halt()* returns true if probing must be stopped for various reasons. These reasons include the detection of a loop or the observation of a gap in the case of a non-responding interface.

Algorithm:

- Procedure Doubletree(h,D)
- B ← φ
- for all d ∈ D do
- h ← AdapthValue (h)
- TraceForward (h,d)
- TraceBackwards (h-1,d)
- End for
- end Procedure
- Procedure AdapthValue(h)
- while ¬ response(v_h) ^ h ≠ 1 do
- h ← h/2
- end while
- Return h
- End procedure
- Procedure TraceForward(i,d)
- While v_i ≠ d ^ (v_i,d) ∉ F ^ ¬halt() do
- F ← F U (v_i,d)
- i ++
- End while
- End procedure
- Procedure Trace Backwards (i,d)
- while i ≥ 1 v_i ∉ B ^ ¬ halt() do
- B ← B U v_i
- F ← F U (v_i,d)
- i –
- End while
- End procedure

Choice of Initial Distance: In order for a monitor to avoid excess intra-monitor redundancy by probing too close and excess inter-monitor redundancy by probing too far, Efficient Doubletree starts off at what is hoped to be an intermediate point *h* between the monitor and the given destination. Each monitor having a different location in the network, a reasonable value for *h* is to be determined for each one.

A choice for *h* is typically based on the distribution of path lengths as seen individually from the perspective of each monitor. One easily estimated parameter by a monitor is its probability *p* of hitting a responding destination on the first probe. By fixing p, the individually obtained values of *h* correspond to a similar level of incursion of each monitor in the network.

Result Analysis: This section describes the simulated version of Efficient Doubletree. The Java code for this prototype is freely available online. This section also describes the results of initial runs of Efficient Doubletree in a real environment.

Instance of Adhoc Network: As an Adhoc network is dynamic and changes time to time, one instance of Adhoc network is given, in the above Figure 4.1.

Path Tracing Information

Forward Probing:: *Global StopSet***:** Towards the Destination1

Monitor1 begins probing from the hop11 and the next hop traced is 16 towards the Destination1. Lines connecting all these nodes can be seen in the network shown in Figure 4.1.

Monitor 2 begins probing from the hop14 and the next hop traced is 18 towards the Destination 1. Lines connecting all these nodes can be seen in the network shown in Figure 4.1.

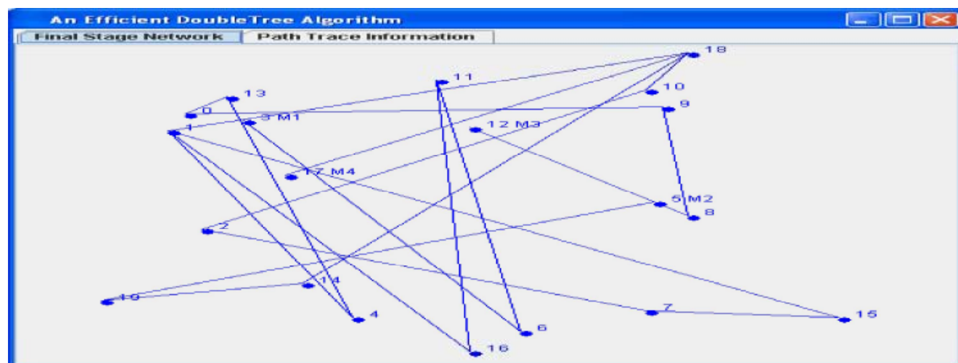

Fig. 4.1: Adhoc Network

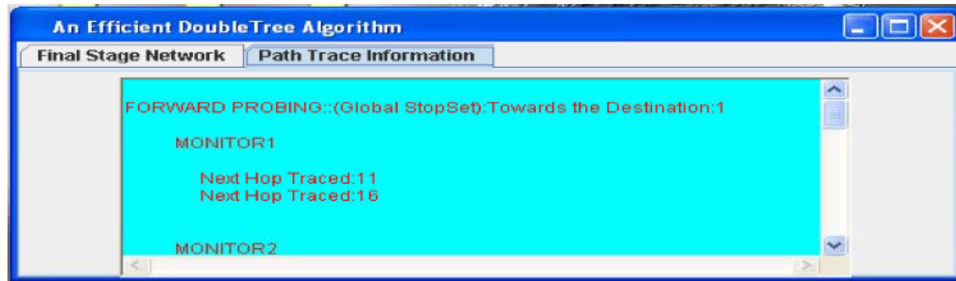

Fig. 4.2: Forward Probing from Monitor1 towards the Destination1

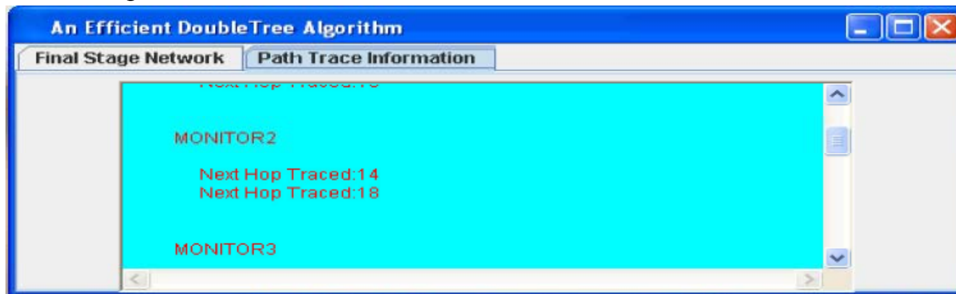

Fig. 4.3: Forward Probing from Monitor2 towards the Destination1

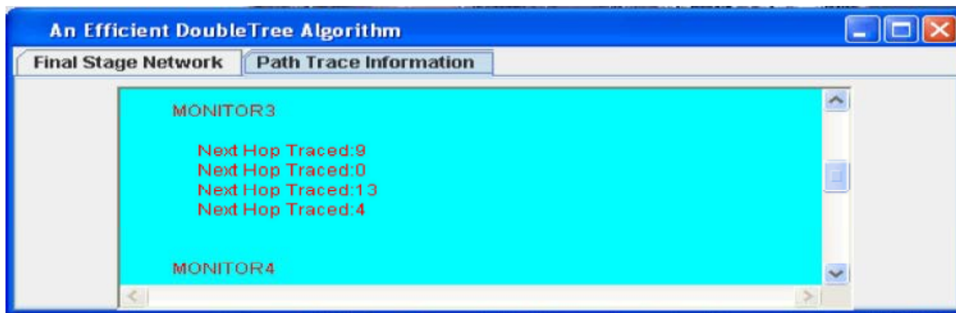

Fig. 4.4: Forward Probing from Monitor3 towards the Destination1

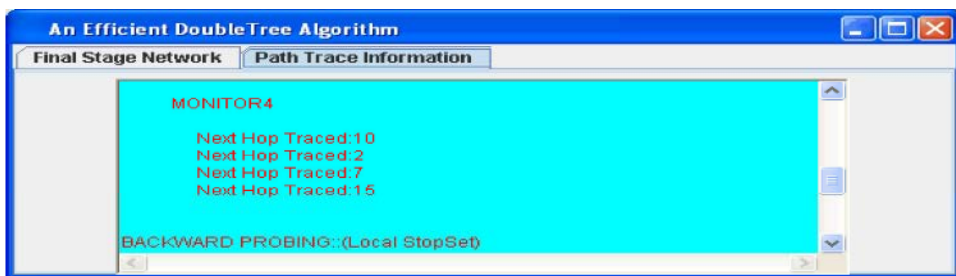

Fig. 4.5 Forward Probing from Monitor4 towards the Destination1

Monitor 3 begins probing from the hop9 and the next hops traced in sequence are 0,13 and 4 towards the Destination1. Lines connecting all these nodes can be seen in the network shown in Figure 4.1.

Monitor 4 begins probing from the hop10 and the next hops traced in sequence are 2, 7 and 15 towards the Destination1. Lines connecting all these nodes can be seen in the network shown in Figure 4.1.

Backward Probing:: *Local StopSet:* Towards the respective Monitors

Backward probing for Monitor1 begins at two-hop distance that is from hop 11 towards hop 6 and Monitor 1 in sequence. Similarly backward probing for Monitor 2 begins at two-hop distance that is from hop14 towards hop19 and Monitor 2 in sequence.

Fig. 4.6: Backward Probing towards Monitor1 and Monitor2 at Two HOP Distance

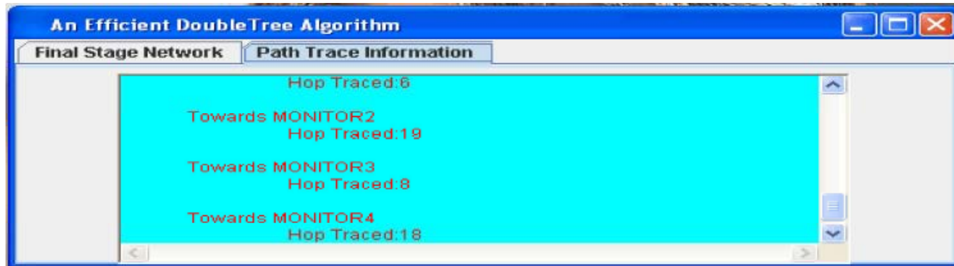

Fig. 4.7: Backward Probing towards Monitor3 and Monitor4 at Two HOP Distance

Backward probing for Monitor3 begins at two-hop distance that is from hop9 towards hop8 and Monitor 3 in sequence. Similarly backward probing for Monitor 4 begins at two-hop distance that is from hop10 towards hop18 and Monitor4 in sequence.

CONCLUSION

In this paper, we quantify the amount of redundancy in classic Internet topology discovery approaches by taking into account the perspective of a single monitor (intramonitor) and that of an entire system (intermonitor). In the intramonitor case, we find that interfaces close to the monitor suffer from a high number of repeat visits. Concerning intermonitor redundancy, we see that a large portion of interfaces are visited by all monitors. In order to scale up classic approaches such as skitter, we have proposed Efficient Doubletree, an algorithm that significantly reduces the duplication of effort, while discovering nearly the same set of nodes and links. Efficient Doubletree simultaneously meets the conflicting demands of reducing intramonitor and intermonitor redundancy. We describe how to tune a single parameter for Efficient Doubletree in order to obtain a tradeoff between redundancy and coverage and we find operating points at which it is possible to strongly reduce one while maintaining the other. For a range of values, Efficient Doubletree is able to reduce measurement load by approximately 76%, while maintaining interface and link coverage above 90%. We also describe a prototype Efficient Doubletree implementation and present preliminary in simulation. These initial results are consistent with the performance expectations formed by the simulation results. Further experiments are necessary, though difficult to carry out. This paper represents the first step towards a highly distributed tracerouting system. Elsewhere, we have proposed extensions to this work: reducing the communication cost of sharing the global stop set through the use of Bloom filters and reducing the load on destinations via techniques we call capping and clustering. In other work, we evaluate and address prefix based stopping rule. We show in this paper that Efficient Doubletree can save time by avoiding duplication of effort between monitors. Therefore, Efficient Doubletree should be able to probe the network more frequently. We beleve that this could make it possible to better capture the network dynamics that result from routing changes and load balancing. Future work might reveal the extent to which Efficient Doubletree is helpful in capturing such network changes. We are also working on a BGP-guided topology discovery tool. We think that a topology discovery algorithm at the IP level may fruitfully make use of information from higher levels, such as the AS-level.

Finally, we plan to work on an overlay or peer-to-peer system for managing the interaction between Efficient Doubletree monitors.

REFERENCES

1. Jacobsen, V., 1989. Traceroute UNIX. [Online]. Available: ftp://ftp.ee.lbl.gov/traceroute.tar.gz, man page.
2. Braden, R., 1989. Requirements for internet hosts. Communication layers, Internet Engineering Task Force, RFC 1122, Oct. [3] E.C. Rosen, “Vulnerabilities of network control protocols: An examples, RFC 789, July 1981.
4. Rekhter, Y., 1991. BGP Protocol Analysis, RFC 1265, October.
5. Atkins, D. and R. Austein, 2004. Threat Analysis of the Domain Name System (DNS). RFC 3833. (Informational), August.
6. Murphy, S., 2006. BGP Security Vulnerable Analysis, RFC 4272 (Informational), January.
7. Retana, A., R. White, V. Fuller and D. McPherson, 2000. Using 31-Bit Prefixes on IPv4 Point-to-Point Links, RFC 3021 (Proposed Standard), December.
8. Rekhter, Y. and T. Li, 1995. A Border Gateway Protocol 4 (BGP-4), RFC 1771 (Draft Standard), March. Obsoleted by RFC 4271.