Evaluation of a Large-Scale Topology Discovery Algorithm

Benoit Donnet[1,2], Bradley Huffaker[2], Timur Friedman[1], and kc claffy[2]

[1] Université Pierre & Marie Curie – Laboratoire LiP6/CNRS, UMR 7606, France
[2] CAIDA – San Diego Supercomputer Center, USA

Abstract. In the past few years, the network measurement community has been interested in the problem of internet topology discovery using a large number (hundreds or thousands) of measurement monitors. The standard way to obtain information about the internet topology is to use the traceroute tool from a small number of monitors. Recent papers have made the case that increasing the number of monitors will give a more accurate view of the topology. However, scaling up the number of monitors is not a trivial process. Duplication of effort close to the monitors wastes time by reexploring well-known parts of the network, and close to destinations might appear to be a distributed denial-of-service (DDoS) attack as the probes converge from a set of sources towards a given destination. In prior work, authors of this paper proposed Doubletree, an algorithm for cooperative topology discovery, that reduces the load on the network, i.e., router IP interfaces and end-hosts, while discovering almost as many nodes and links as standard approaches based on traceroute. This paper presents our open-source and freely downloadable implementation of Doubletree in a tool we call traceroute@home. We evaluate the performance of our implementation on the PlanetLab testbed and discuss a large-scale monitoring infrastructure that could benefit of Doubletree.

1 Introduction

Today's most extensive tracing system at the IP interface level, *skitter* [1], uses 24 monitors, each targeting on the order of one million destinations. Authors of this paper are responsible for skitter. In the fashion of skitter, *scamper* [2] makes use of several monitors to traceroute IPv6 networks. Other well known systems, such as RIPE *NCC TTM* [3] and NLANR *AMP* [4], each employ a larger set of monitors, on the order of one- to two-hundred, but they avoid probing outside their own network. However, recent work has indicated the need to increase the number of traceroute sources in order to obtain a more complete topology measurement [5, 6]. Indeed, it has been shown that reliance upon a relatively small number of monitors to generate a graph of the internet can introduce unwanted biases.

One way of rapidly creating a large distributed monitor infrastructure would be to deploy traceroute monitors in an easily downloadable and readily usable

piece of software, such as a screensaver. This was first proposed by Jörg Nonnenmacher, as reported by Cheswick et al. [7]. Such a suggestion is in keeping with the spirit of that have arisen in the past few years. The most famous one is probably SETI@home [8]. SETI@home's screensaver downloads and analyzes radio-telescope data. The first publicly downloadable distributed route tracing tool is DIMES [9], released as a daemon in September 2004. At the time of writing this paper, DIMES counts more than 8,700 agents scattered over five continents.

However, building such a large structure leads to potential scaling issues: the quantity of probes launched might consume undue network resources and the probes sent from many vantage points might appear as a distributed denial-of-service (DDoS) attack to end-hosts. These problems were quantified in our prior work [10].

The Doubletree algorithm [10], proposed by two authors of this paper, is an attempt to perform large-scale topology discovery efficiently and in a network friendly manner. Doubletree acts to avoid retracing the same routes in the internet by taking advantage of the tree-like structure of routes fanning out from a source or converging on a destination. The key to Doubletree is that monitors share information regarding the paths that they have explored. If one monitor has already probed a given path to a destination then another monitor should avoid that path. Probing in this manner can significantly reduce load on routers and destinations while maintaining high node and link coverage [10]. By avoiding redundancy, not only is Doubletree able to reduce the load on the network but it also allows one to probe the network more frequently. This makes it possible to better capture network dynamic (routing changes, load balancing) compared to standard approaches based on traceroute.

This paper goes beyond earlier theory and simulation to propose a Doubletree implementation in tool called *traceroute@home*. traceroute@home is open-source and freely available [11][3]. The goal of this paper is neither to compare Doubletree to standard probing (e.g., skitter) in a real environment, neither to tell the story of the large-scale deployment of Doubletree. We aim to evaluate our implementation of the algorithm and understand its behavior in a real, but controlled, environment, i.e., PlanetLab [13]. This approach can be seen as a first step towards a larger-scale deployment of the algorithm in an entirely dedicated measurement structure. We first implement and evaluate the core of the measurement system, i.e., the probing engine, before building a more complex infrastructure. Our implementation is modular, making future extensions and reuse in a dedicated measurement structure easy. In this paper, we also discuss a large-scale measurement infrastructure that could benefit of traceroute@home.

The remainder of this paper is organized as follows: Sec. 2 describes the Doubletree algorithm; Sec. 3 describes traceroute@home; in Sec. 4, we discuss the performance evaluation done on PlanetLab nodes; Sec. 5 discusses the usage of traceroute@home in an entirely dedicated measurement infrastructure; finally, Sec. 6 summarizes the principal contributions of this paper.

[3] Interested readers might find an extended version of this paper in an arXiv technical report [12].

2 Doubletree

Doubletree [10] is the key component of a coordinated probing system that significantly reduces load on routers and end-hosts while discovering nearly the same set of nodes and links as standard approaches based on traceroute. It takes advantage of the tree-like structures of routes in the context of probing. Routes leading out from a monitor towards multiple destinations form a tree-like structure rooted at the monitor. Similarly, routes converging towards a destination from multiple monitors form a tree-like structure, but rooted at the destination. A monitor probes hop by hop so long as it encounters previously unknown interfaces. However, once it encounters a known interface, it stops, assuming that it has touched a tree and the rest of the path to the root is also known. Using these trees suggests two different probing schemes: backwards (monitor-rooted tree) and forwards (destination-rooted tree).

For both backwards and forwards probing, Doubletree uses stop sets. The one for backwards probing, called the *local stop set*, consists of all interfaces already seen by that monitor. Forwards probing uses the *global stop set* of (interface, destination) pairs accumulated from all monitors. A pair enters the stop set if a monitor received a packet from the interface in reply to a probe sent towards the destination address.

A monitor that implements Doubletree starts probing for a destination at some number of hops h from itself. It will probe forwards at $h + 1$, $h + 2$, etc., adding to the global stop set at each hop, until it encounters either the destination or a member of the global stop set. It will then probe backwards at $h - 1$, $h - 2$, etc., adding to both the local and global stop sets at each hop, until it either has reached a distance of one hop or it encounters a member of the local stop set. It then proceeds to probe for the next destination. When it has completed probing for all destinations, the global stop set is communicated to the next monitor.

Doubletree has one tunable parameter. The choice of initial probing distance h is crucial. Too close, and duplication of effort will approach the high levels seen by classic forwards probing techniques [10, Sec. 2]. Too far, and there will be high risk of traffic looking like a DDoS attack for destinations. The choice must be guided primarily by this latter consideration to avoid having probing look like a DDoS attack.

While Doubletree largely limits redundancy on destinations once hop-by-hop probing is underway, its global stop set cannot prevent the initial probe from reaching a destination if h is set too high. Therefore, each monitor sets its own value for h in terms of the probability p that a probe sent h hops towards a randomly selected destination will actually hit that destination. Fig. 2 shows the cumulative mass function for this probability for skitter monitor `champagne`. If one considers as reasonable a 0.2 probability of hitting a responding destination on the first probe, it must chose $h \leq 14$.

Simulation results [10, Sec. 3.2] show for a range of p values that, compared to classic probing, Doubletree is able to reduce measurement load by approximately 70% while maintaining interface and link coverage above 90%.

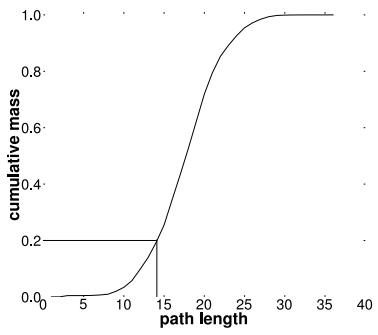


Fig. 1. Cumulative mass plot of path lengths from skitter monitor `champagne`

Doubletree assumes, in the context of probing, that the routes have a tree-like structure. This is true in a large proportion, as suggested by Doubletree's coverage results (see [10, Sec. 3.2]), but this hypothesis implies a static view of the network. When a Doubletree monitor stops probing towards the root of a tree, it is making the bet that the rest of the path to the tree is both know and unchanged since earlier probing. The existence of routes' convergence and divergence points, however, imply a dynamic view of the network, as some parts of the network might change due to load balancing and routing. We are currently working on improving Doubletree in order to take into account dynamic behaviors of the network [14].

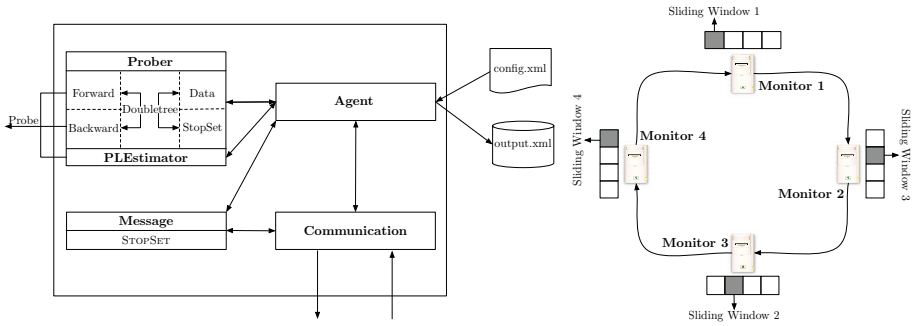
3 Implementation

In this section, we describe our implementation of the Doubletree algorithm in a tool called traceroute@home. We first introduce our design choices (Sec. 3.1) and, next, we give an overview of the system (Sec. 3.2).

3.1 Design Choices

We implemented traceroute@home in Java [15]. We choose Java as the development language because of two reasons: the large quantity of available packages and the possibility of abstracting ourselves from technical details. As a consequence, the development time was strongly reduced. Unfortunately, Sun does not provide any package for accessing packet headers and handling raw sockets, which is necessary to implement traceroute. Instead of developing our own raw sockets library, we used the open-source *JSocket Wrench* library [16]. We modified the JSocket Wrench library in order to support multi-threading. Our modifications are freely available [11].

We aimed for the design of traceroute@home to be easily extended in the future by ourselves but also by the networking community. For instance, con-



(a) A traceroute@home monitor

(b) Doubletree with sliding window

Fig. 2. The traceroute@home system

cerning the messages exchanged by monitors, we define a general framework for messages, making creation and handling of new messages easier.

We designed our application by considering two levels: the *microscopic* level and the *macroscopic* level.

From a macroscopic point of view, i.e., all the monitors together, the monitors are organized in a ring, adopting a round robin process. At a given time, each monitor focuses on its own part of the destination list. When it finishes probing its part, it sends information to the next monitor and waits for data from the previous one, if it was not yet received. Sec. 3.2 explains this macroscopic aspect of traceroute@home.

From a microscopic point of view of our implementation, i.e., a single monitor, a monitor is tuned with an XML configuration file loaded at its starting. A traceroute@home monitor is composed of several modules that interact with each other. Our implementation is thread-safe, as a monitor, conducted by the *Agent* module, is able to send several probes (ICMP or UDP) and receive several messages from other monitors at the same time. Further, topological information collected by a monitor is regularly saved to XML files. Obviously, a traceroute@home monitor implements a *Doubletree* module, as described in Sec. 2. Fig. 2(a) illustrates a traceroute@home monitor. Going into deeper details within a traceroute@home monitor is beyond the scope of this paper. Nevertheless, interested readers might find a complete description of a traceroute@home monitor in [12].

Our implementation is open-source and freely available [11].

3.2 System Overview

The simulations conducted in prior work [10] were based on a simple probing system: each monitor in turn covers the destination list, adds to the global stop set the (interface, destination) pairs that it encounters, and passes the set to the subsequent monitor.

This simple scenario is not suitable in practice: it is too slow, as an iterative approach allows only one monitor to probe the network at a given time. We want all the monitors probing in parallel. However, how would one manage the global stop set if it were being updated by all the monitors at the same time?

An easy way to parallelize is to deploy several *sliding windows* that slide along the different portions of the destination list. At a given time, a given monitor focuses on its own window, as shown in Fig. 2(b). There is no collision between monitors, in the sense that each one is filling in its own part of the global stop set. The entire system counts m different sliding windows, where m is the number of Doubletree monitors. If there are n destinations, each window is of size $w = n/m$. This is an upper-bound on the window size as the concept still applies if they are smaller.

A sliding window mechanism requires us to decide on a step size by which to advance the window. We could use a step size of a single destination. After probing that destination, a Doubletree monitor sends a small set of pairs corresponding to that destination to the next monitor, as its contribution to the global stop set. It advances its window past this destination, and proceeds to the next destination. Clearly, though, a step size of one will be costly in terms of communication. Packet headers (see [12] for details about packet format) will not be amortized over a large payload, and the payload itself, consisting of a small set, will not be as susceptible to compression as a larger set would be.

On the other hand, a step size equal to the size of the window itself poses other risks. Suppose a monitor has completed probing each destination in its window, and has sent the resulting subset of the global stop set on to the following monitor. It then might be in a situation where it must wait for the prior monitor to terminate its window before it can do any further useful work.

A compromise must be reached, between lowering communications costs and continuously supplying each monitor with useful work. This implies a step size somewhere between 1 and w. For our implementation of Doubletree, we let the user decide the step size. This is a part of the XML configuration file that each Doubletree monitor loads at its start-up (see Fig. 2(a) and [12]). Future work might reveal information about how to tune the step size of a monitor.

4 Performance Evaluation

As described in prior work [10], security concerns are paramount in large-scale active probing. It is important to not trigger alarms inside the network with Doubletree probes. It is also important to avoid burdening the network and the destination hosts. It follows from this that the deployment of a cooperative active probing tool must be done carefully, proceeding step by step, from an initial small size, up to larger-scales. Note that this behavior is strongly recommended by PlanetLab [17, Pg. 5].

We tested traceroute@home on a set of ten PlanetLab nodes. These ten nodes acted as traceroute@home monitors. We selected them based on their relatively high stability (i.e., remaining up and connected), and their relatively low load.

	Backwards				Forwards				
monitor	loop	gap	stop set	normal	loop	gap	stop set	normal	**h**
`Blast`	0	0	99.5	0.5	2	17	50	31	7
`Cornell`	0	0	99	1	0	13.5	69.5	17	7
`Ethz`	1	0	98.5	0.5	2	10.5	52	35.5	11
`Inria`	1.5	0	97.5	1	1	4	67	28	15
`Kaist`	0	0	99	1	0.5	10.5	64.5	24.5	9
`Nbgisp`	0.5	4	95	0.5	3.5	30.5	22	44	7
`LiP6`	0	0	99.5	0.5	1	9.5	62.5	27	11
`UCSD`	0	0	99.5	0.5	0	10.5	60.5	29	7
`Uoregon`	0	0	99.5	0.5	0	7	74.5	18.5	6
`Upc`	0.5	0	99	0.5	1	14	57.5	27.5	15
mean	0.35	0.4	98.6	0.65	0.11	12.7	58	28.2	9

Table 1. Stopping reasons (in %) and h value per monitor

These traceroute@home monitors are scattered around the world: North America (USA, Canada), Europe (France, Spain, Switzerland, Spain), and Asia (Japan, Korea). Evaluating the performance of selected PlanetLab nodes is beyond the scope of this paper. Interested readers might find further information about such an evaluation in [12].

The destination list, i.e., the probe targets consists of $n = 200$ PlanetLab nodes randomly chosen amongst the approximately 300 institutions that currently host PlanetLab nodes. Restricting ourselves to PlanetLab nodes destinations was motivated by security concerns. By avoiding tracing outside the PlanetLab network, we avoid disturbing end-systems that do not welcome probe traffic. None of the ten PlanetLab monitors (or other nodes located at the same place) belongs to this destination list. The sliding window size of $w = n/m$ consists of twenty destinations. We consider two step sizes by window, so each step counts ten destinations. Each traceroute@home monitor was configured as follows: the probability p was set to 0.05, no compression was required before sending messages and a sliding window was composed of two step sizes.

The experiment was run on the PlanetLab nodes on Dec. 20$^{\text{th}}$ 2005. All the traceroute@home monitors were started at the same time. The experiment was finished when each monitor had probed the entire destination list.

A total of 2,703 links and 2,232 nodes were discovered. We also encountered 2,434 non-responding interfaces (routers and destinations). We recorded 36 invalid addresses. Invalid addresses are, for example, private addresses [10, Sec. 2.1].

Table 1 shows the different reasons for stopping backwards and forwards probing for each traceroute@home monitor. It further indicates the h value computed by each monitor. The last row of the table indicates the mean for each column. A *loop* occurs when a given node appears at two different hops. A *gap* occurs when five successive nodes does not reply to probes. A *stop set* indicates the application of a stopping rule based on the membership to a given stop

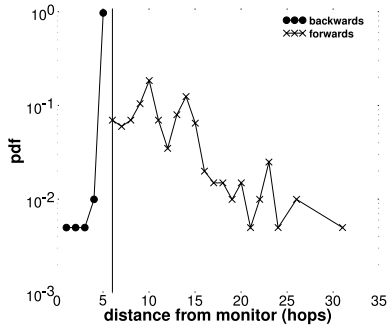


Fig. 3. Stopping distance for the `Uoregon` monitor

set (local stop set for backwards and global stop set for forwards), as defined in Sec. 2). A *normal* stopping means hitting the first hop (backwards) or the destination (forwards).

Looking first at the backwards stopping reasons, we see that the stop set rule strongly dominates (98.6% on average). On average, normal stopping occurs only 0.65% of the time.

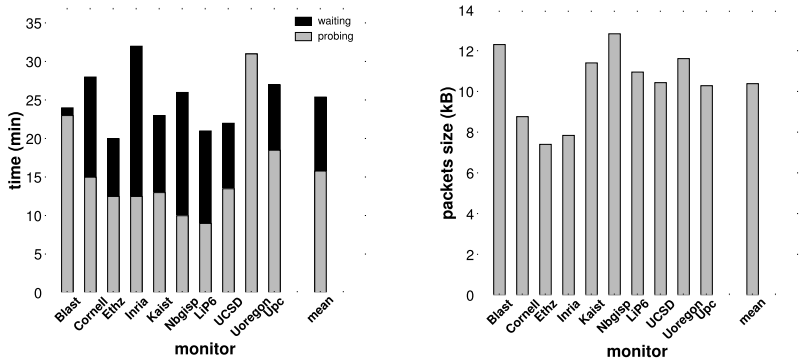
Fig. 3 shows the stopping distance (in terms of hops from the monitor), for the `Uoregon` monitor, when probing backwards and forwards. The vertical line indicates the h value computed by `Uoregon`. Results presented in Fig. 3 are typical for the other traceroute@home monitors.

We see that more than 90% of the backwards stopping occurs at a distance of 5, that is to say the distance corresponding to $h - 1$. In 2.5% of the cases, the probing stops between hop 1 (normal stopping) and hop 4. Except for hop 1, the other stops might be caused by the stop set or by hitting a destination, probably due to very short paths. This latter case illustrates a situation in which the first probe sent with a TTL of h directly hits a destination.

Looking now at the forwards stopping reasons in Table 1, we see that the gap rule plays a greater rule. We believe that these gaps occur when a destination does not respond to probes because of a restrictive firewall or because the PlanetLab node is down.

On average, in 58% of the cases, the stop set rule applies, and in 28.2% of the cases, the normal rule applies. The normal rule proportion might be seen as high but we have to keep in mind that a Doubletree monitor starts with an empty stop set. Therefore, during the first sliding window, the only thing that can stop a monitor, aside from the gap rule, is an encounter with the destination.

Looking at the stopping distance in Fig. 3, we see that the distances are more scattered for forwards probing than for backwards probing. Regarding the forwards probing, a peak is reached at a distance of 10 (18.5% of the cases). In 7% of the cases, the monitor stops probing at a distance of 6, that is equal to the value h. It could correspond to the stop set rule application or the normal rule, by definition of p. Recall that p defines the probability of hitting a destination with



(a) running time (b) volume sent

Fig. 4. Load

the probe sent with a TTL equals to h. For our experiment, we set $p = 0.05$, meaning that in 5% of the cases the first probe sent by a monitor will hit a destination.

Fig. 4 shows, for each traceroute@home monitor, the load generated by our prototype. This load is expressed in terms of the running time (Fig. 4(a)) and of the total size of packets exchanged by monitors (Fig. 4(b)). Each figure has an additional bar on the right of the plot that gives the mean over the ten monitors.

The size of packets takes into account the header (4 bytes) and the payload. Interested readers might find further information about messages in [12]. The messages exchanged by monitors are StopSet messages. A StopSet message is sent by a monitor when it reaches a step size in the current sliding window and contains stop set information for the next monitor in the ring (See Fig. 2(b)). As we define for our experiment two step sizes per sliding window and as we deploy our prototype on ten PlanetLab nodes, each monitor sent 20 StopSet messages.

The monitors do not exchange their entire stop set. They only send an update that contains the (interface, destination) pairs discovered during the current step size probing.

In Fig. 4(b), we can see that a monitor sends a total of between 7.41 KB and 12.84 KB to the subsequent monitor. On average, a monitor sends 10.39 KB of stop set information into the network.

During our experimentation, the traceroute@home application did not flood the network with StopSet messages. However, our prior work [18] has shown, on a larger destination list, that it can grow to excessive sizes. In this case, our prior work suggests to implement the global stop set as a Bloom filter [18] instead of a list of (interface, destination) pairs. This implementation is provided in our prototype. Our prototype is easily tunable, due to the use of an XML configuration file. The user must specify in this file which type of implementation the prototype has to use. For our experiments, we choose to consider the standard implementation of the global stop set, i.e., the list.

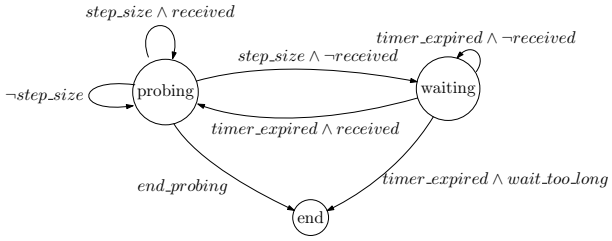


Fig. 5. Probing/waiting state interactions

Looking now at the running time (Fig. 4(a)), we see that it is expressed as a combination of probing (gray bars) and waiting periods (black bars). The waiting period occurs when a monitor has finished its sliding window or a step size in a given sliding window and is waiting for the global stop set that should be sent by the previous monitor in the round-robin topology. We see that nearly all monitors have to wait. A waiting period, in our implementation, lasts 30 seconds. When the timer expires, the monitor checks if it received a new message. If so, the waiting period ends and a new probing period begins. Otherwise, it sleeps during 30 seconds. To avoid infinite waiting, if after 40 sleeping periods (i.e., 20 minutes), nothing was received, the monitor quits with an error message. Fig. 5 illustrates the interactions between the probing state and the waiting state.

We believe that these long waiting periods are due to a characteristic of the PlanetLab IP stack. It seems that when ICMP replies are by the stack, the socket reader function does not read them immediately. As the timer set on the listening socket never expires in this case, we think that the socket reader function is waiting for the permission to access the IP stack. It looks like the resource is owned (or locked) by another process on the PlanetLab node. Note that this behavior was also noticed by other Planet-Lab users [19].

5 Measurement Infrastructures

The recent NSF-sponsored CONMI Workshop [20] (in which two of the present authors participated) urged a comprehensive approach to distributed probing, with a shared infrastructure that respects the many security concerns that active measurements raise. We echo this call and believe that Doubletree falls within the scope of the trade-off between probing load and the information gleaned from such probing. In this section, we discuss a brand new infrastructures that could take advantage of Doubletree and, thus, traceroute@home.

OneLab [21] is a European project, due to start in September 2006, that assembles some of the most highly respected network research teams from university and industry labs in Europe. OneLab aims to extend PlanetLab into new environments beyond the traditional wired internet, to deepen PlanetLab's monitor capabilities, and to provide a European administration for PlanetLab nodes in Europe.

OneLab's monitoring component is mainly motivated by the fact that many applications, such as those that take advantage of multihoming, could benefit from a better vision of the characteristics of the underlying network. Some objectives of the monitoring component are designing and implementing a prototype measurement infrastructure providing router and AS-level path information. The project also intends to submit the definition of a standard API for the measurement platform to the IETF (or IRTF).

This infrastructure has the potential to perform large-scale measurements. That is why we believe that Doubletree and, by extension, traceroute@home would perfectly fit into these plans. Further, we designed our implementation while keeping in mind extensibility. Changes or extensions needed when incorporating our prototype within the active measurement monitoring component of OneLab will be easy to achieve. In addition, due to the use of XML, the prototype is easy to tune and the resulting topological information might be quickly changed in an other format than XML if needed.

6 Conclusion

In this paper, we were interested in large-scale topology discovery at IP level. We focused on Doubletree, an efficient and cooperative topology discovery algorithm.

We put Doubletree one step further than its initial description by proposing a Java implementation. The application that implements Doubletree is called traceroute@home, is easily extensible, is open-source and freely available.

We discussed our implementation by explaining our design choices and by presenting the global functioning of the system. We next evaluated the performance of traceroute@home and described its behavior in a real environment. We finally discussed a monitoring infrastructure that could benefit of traceroute@home.

traceroute@home is an on-going project. We aim to improve our tool. In the near future, we would like to develop IPv6 libraries in order to permit IPv6 networks probing. Further, we are currently developing a peer-to-peer (or an overlay) system for managing the probing monitors and the entire structure. This new architecture is based on the prototype discussed in this paper.

Acknowledgements

Mr. Donnet's work was supported by a SATIN European Doctoral Research Foundation grant and by an internship at CAIDA.

References

1. Huffaker, B., Plummer, D., Moore, D., claffy, k.: Topology discovery by active probing. In: Proc. Symposium on Applications and the Internet. (2002)

2. Luckie, M.: IPv6 scamper (2005) WAND Network Research Group.
3. Georgatos, F., Gruber, F., Karrenberg, D., Santcroos, M., Susanj, A., Uijterwaal, H., Wilhelm, R.: Providing active measurements as a regular service for ISPs. In: Proc. Passive and Active Measurement (PAM) Workshop. (2001)
4. McGregor, A., Braun, H.W., Brown, J.: The NLANR network analysis infrastructure. IEEE Communications Magazine **38** (2000)
5. Lakhina, A., Byers, J., Crovella, M., Xie, P.: Sampling biases in IP topology measurements. In: Proc. IEEE INFOCOM. (2003)
6. Clauset, A., Moore, C.: Traceroute sampling makes random graphs appear to have power law degree distributions. cond-mat 0312674, arXiv (2004)
7. Cheswick, B., Burch, H., Branigan, S.: Mapping and visualizing the internet. In: Proc. USENIX Annual Technical Conference. (2000)
8. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@home: An experiment in public-resource computing. Communications of the ACM **45** (2002)
9. Shavitt, Y., Shir, E.: DIMES: Let the internet measure itself. ACM SIGCOMM Computer Communication Review **35** (2005)
10. Donnet, B., Raoult, P., Friedman, T., Crovella, M.: Efficient algorithms for large-scale topology discovery. In: Proc. ACM SIGMETRICS. (2005)
11. Donnet, B.: traceroute@home 1.0 (2006) See `http://trhome.sourceforge.net`.
12. Donnet, B., Huffaker, B., Friedman, T., claffy, k.: Implementation and deployment of a distributed network topology discovery algorithm. cs.NI 0603062, arXiv (2006)
13. PlanetLab Consortium: PlanetLab project (2002) See `http://www.planet-lab.org`.
14. Donnet, B., Huffaker, B., Friedman, T., claffy, k.: Increasing the coverage of a cooperative internet topology discovery algorithm (2006) Under review.
15. Microsystems, J.S.: JDK 1.4.2 (1994) `http://java.sun.com`.
16. JSocket Wrench: Release R04 (2004) See `http://jswrench.sourceforge.net/`.
17. Peterson, L., Pai, V., Spring, N., Bavier, A.: Using PlanetLab for network research: Myths, realities, and best practices. Design Note PDN–05–028, PlanetLab Consortium (2005)
18. Donnet, B., Friedman, T., Crovella, M.: Improved algorithms for network topology discovery. In: Proc. Passive and Active Measurement (PAM) Workshop. (2005)
19. Planet-Lab users mailing-list: Very high ping/traceroute latencies on planet-lab nodes (2006) `http://lists.planet-lab.org/pipermail/users/2006-March/001892.html`.
20. claffy, k., Crovella, M., Friedman, T., Shannon, C., Spring, N.: Community-oriented network measurement infrastructure (CONMI) (2005) Workshop Report. Available from `http://www.caida.org/publications/papers/2005/conmi`.
21. OneLab consortium headed by Université Pierre & Marie Curie: The onelab project (2006) See `http://www.one-lab.org`.