DTS: a Decentralized Tracing System

Kenji Masui[‡] and Benoit Donnet[†] [⋆]

[‡] Tokyo Institute of Technology `kmasui@gsic.titech.ac.jp`
[†] Université catholique de Louvain `benoit.donnet@uclouvain.be`

Abstract. A new generation of widely distributed systems to measure the Internet topology at the interface level is currently being deployed. Cooperation between monitors in these systems is required in order to avoid over-consumption of network resources. This paper proposes an architecture for a distributed topology measurement (DTM) system that, for the first time, decentralizes probing information. The key idea of our proposal is that, by utilizing a shared database as a communication method among monitors and taking advantage of the characteristics of the Doubletree algorithm, we can get rid of a specific control point, and a DTM system can be constructed in a decentralized manner. In this paper, we describe our implementation of a DTM, called Decentralized Tracing System (DTS). Decentralization within DTS is achieved using various distributed hash tables (DHTs), each one being dedicated to a particular plane (i.e., control or data). We also provide preliminary evaluation results.

1 Introduction

The past ten years have seen a growing body of important research work on the Internet topology [1]. The work is based on maps built by systems such as *Archipelago* [2], probing the Internet topology from multiple vantage points using the technique of *traceroute*. We call these *distributed topology measurement* (*DTM*) systems. Large-scale DTM systems are attracting researchers' attention due to their better capabilities of tracking network dynamics. Given we have more number of monitors for probing specific networks, each monitor can take a smaller portion of the topology and probe it more frequently. Changes that might be missed by smaller systems can more readily be captured by the larger ones, while keeping the workload per monitor constant. 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 on end-hosts [3, 4]. The NSF-sponsored CONMI Workshop [5] urged a comprehensive approach to distributed probing, with a shared infrastructure that respects the many security concerns that active measurements raise. DTMs must coordinate the efforts of their individual monitors.

In this paper, we propose the first decentralized architecture for a DTM, called *Decentralized Tracing System* (DTS). We build on our prior work [3] in introducing cooperation among tracing monitors, through the *Doubletree* topology discovery algorithm. Doubletree takes advantage of the tree-like structure of

[⋆] Benoit Donnet is funded by the Fonds National de la Recherche Scientifique (FNRS – Rue d'Egmont 5, 1000 Brussels).

routes, either emanating from a single source to multiple destinations or routes converging from multiple sources to a single destination, in order to avoid duplication of effort for topology discovery. With Doubletree, tracing monitors cooperate by exchanging information about which interfaces were previously discovered through probing specific interfaces. Doubletree describes what must be shared but, prior to this work, we did not specify precisely how it should be shared in a distributed environment.

Our DTS makes use of a storage built on the technique of distributed hash table (DHT) for decentralizing its *control* and *data* planes. Moreover, because of the uncertain environment that DTMs must run in, where host machines are susceptible to varying network load and possible disconnection, they require an architecture that is not just scalable, but is also flexible and robust. We also consider these matters on designing DTS (Sec. 2), and discuss the preliminary evaluation of DTS (Sec. 3). Our implementation is freely available.[1]

2 Design and Implementation of DTS

2.1 DTM Systems Requirements

Control Plane The *control plane* of a DTM system refers to the management of information regarding probing targets as well as information needed to decide when probing must stop for a given target.

First, a DTM system has to share the target list, i.e., the list of IP addresses (or names) of probe targets, between probing monitors. A target list must be permanent in the system. However, one must have the opportunity to perform on the fly some changes in the list, such as adding or removing items. For instance, a target can refuse to be probed in the future and its IP address must be then blacklisted and removed from the current target list. For the rest of this paper, we refer to the target list as *probing target* (PT).

Second, a DTM system has to share information to guide probing in order to make measurements more efficient. This information can help a probing monitor to decide when to stop probing a given target. By definition, such an information is volatile. In the following, we refer to this information as *probing control information* (PC).

A DTM system must be *dynamic.* It should accept dynamic arrivals and departures (volunteer or not) of monitors. Monitors join and leave the system when they wish (*flexibility*). Such a dynamic behavior must have limited impact on the shared information (*robustness*).

Finally, the control plane of a DTM system must ensure that each probing monitor can perform measurements at its own pace. A slower monitor cannot slowdown others monitors, which is another property for *flexibility.*

Data Plane The *data plane* of a DTM system refers to the topological data collected during probing. In the fashion of the Archipelago data, the result data set should be accessible by the research community. A DTM system has to keep

[1] See `http://www.n-tap.net/`

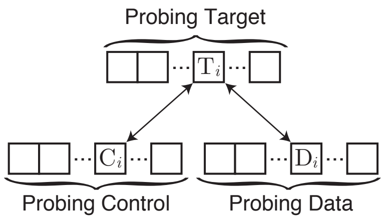

Fig. 1. Relationship between shared information.

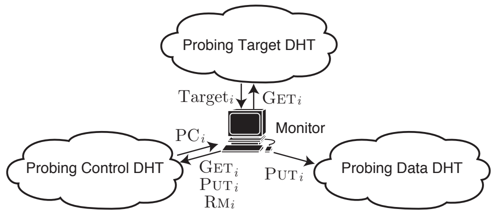

Fig. 2. DTS and the dedicated DHTs.

track of each probing result, for each probing monitor, from the beginning and must ensure the long-term persistence of this data set.

The DTM system must provide an easy access to the data storage system. On the one hand, probing monitors must be able to efficiently and easily store the data collected so that the whole system avoids a bottleneck issue when storing data. On the other hand, the information must be easily retrieved for research purposes. In the following, we refer to the collected data as *probing data* (*PD*).

2.2 Design and Implementation

In this section, we describe the *Decentralized Tracing System* (DTS), the first entirely distributed topology discovery system, and explain how our implementation meets the requirements provided in Sec. 2.1.

Previous works on Internet topology discovery include, among others, *DIMES* [6] (publicly released as a daemon), *Rocketfuel* [7] (focusing on the topology of a given ISP), *Scriptroute* [8] (a system that allows an ordinary Internet user to perform network measurements from several distributed vantage points), and *iPlane* [9] (construction of an annotated map of the Internet). All of these systems operate under central control. Indeed, unlike DTS, Rocketfuel and Scriptroute assume a centralized server to share stopping information (i.,e., the list of previously observed IP addresses). Rocketfuel and Scriptroute do not consider how the information regarding where to stop probing can be efficiently encoded for exchange between monitors.

Global View of DTS In Sec. 2.1, we explained that a DTM system has to share information for controlling probing but also for managing the data. DTS, our implementation of a DTM system, requires three information to be shared among monitors: the probing control information, the probing target, and the probing data.

Sharing probing target and probing control information between a large set of monitors might lead to scaling issues. For instance, it could be a problem if all the monitors try to access the probing control information (or a particular item of the probing control information) at the same time. Further, if all the monitors probe the entire destination list at the same time, it is difficult to benefit from work performed by others and, consequently, difficult to exchange probing control information. A way to avoid such a problem would be to divide the target list into *chunks*. A chunk is a portion of the entire target list and there is no overlapping between chunks. Each monitor focuses, at a given time, on its own chunk. To each probing target chunk is associated a specific probing information chunk and a specific probing data chunk. Fig. 1 illustrates the relationship between a specific probing target chunk, T_i, the related information used to guide probing, C_i, and the topological data collected by monitors, D_i.

The key idea of DTS is to enable communication between monitors through the use of DHTs. For any information to share, DTS employs a dedicated DHT. Given that each DTS monitor has to share three information, the whole system requires three different DHTs, as depicted in Fig. 2.

Each value stored by a specific DHT refers to a chunk. For instance, the Probing Target DHT on Fig. 2 stores target chunks. Further, a key in a DHT will serve as the identifier for a particular chunk. For consistency reasons, the key for a target chunk is the same that the key for the corresponding probing information and data. To this end, a number is associated to each chunk and the key of the chunk is calculated based on this number.

Control Plane The control plane of DTS is composed of several modules that interact through the *Agent* engine.

A DTS monitor probes the network with its *Prober* engine, which implements the Doubletree algorithm that is based on both backwards and forwards probing as well as the stop sets [3]. The control plane of DTS interacts with the PC DHT in order to store and retrieve the stop set corresponding to the current chunk.

Our approach in constructing DTS is somewhat similar to Chawathe et al. [10] who evaluate whether it is possible to use DHTs as an application-independent building block to implement a key component of an end-user positioning system. DTS is a DTM system that makes use of DHTs to share information between monitors. One of the key ideas we had in mind when designing DTS was its ease of deployment. We therefore choose to make DTS free from DHT specifications. Instead, we provide a *DHT Abstraction* engine, making the DHT transparent to a monitor as it interacts only with the DHT Abstraction. In particular, the DHT Abstraction engine interacts with the interfaces provided by N-TAP [11]. These interfaces allow other systems to utilize the features of N-TAP including the shared database and communication channels among monitors. The DHT Abstraction engine converts the information that are exchanged between the control plane and N-TAP so that it can provide consistent interfaces to other modules in DTS.

Data Plane Our implementation of the data plane is somewhat similar to the control plane. The difference stands in the fact that the Prober engine is replaced by a *Data* engine. The objective of the Data engine is to transform the raw replies (i.e., ICMP received) into well formatted data that contains additional information useful for the research community, such as timestamps, stopping reasons, DTS monitor name, chunk identifier, etc.. The collected data is, then, sent through the DHT abstraction to the PD DHT.

Adaptation to N-TAP According to the design presented so far, we describe how DTS is implemented on an existing measurement platform, N-TAP [11].

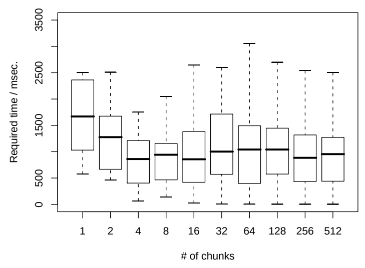

Fig. 3. Required time for retrieving one chunk from PT DHT.

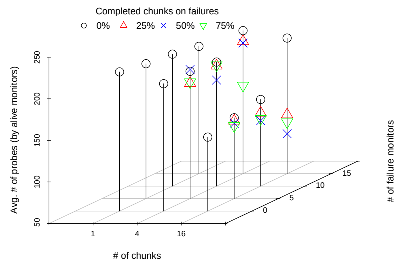

Fig. 4. Impact of the failure of monitors and the chunk size on the number of probes.

Basically, the N-TAP platform consists of *N-TAP agents* that are assumed to reside in multiple administrative domains. Besides the agents perform measurement, the agents also play a role in forming a measurement overlay network with the technique of Chord [12]. The overlay network is called the *N-TAP network*, which provides some high-level functions such as shared database among agents. In N-TAP, there are two roles of agents: *core* and *stub* [13]. The core agents have to maintain a Chord-based peer-to-peer network for its DHT service, meanwhile, the stub agents do not need to maintain the network but join the network via a core agent. These two kinds of agents form a bi-layered peer-to-peer network.

For constructing the DTS, we prepare several stable nodes as core agents that can serve the shared database. Since the number of the core agents has an impact on the scalability of DTS, we should carefully choose the number. On the other hand, in principle, DTS monitors play a role of a stub agent and do not engage in the maintenance of the DHT service. The monitors, of course, perform topology discovery based on the Doubletree algorithm, and can utilize the dedicated DHTs (for PC/PT/PD) via a core agent. Briefly, core agents work as a gateway of the DHT service for stub agents.

3 Evaluation

The decentralized architecture of DTS, which is based on the DHT-based storage and the intercommunication among monitors via the storage (not the direct communication among the monitors), provides some advantages. We can summarize them with four points: flexibility, robustness, scalability, and modularity. The robustness in DTS is related to the impact of monitor failures. When a monitor (or several monitors) fails, the entire system must continue to work. Further, the information lost (probing data and probing control) due to the failure must be limited. In this section, we evaluate the robustness of DTS through the impact of the chunk size and monitor failure.

Even though DTS can maintain its function, the failure of monitors causes the loss of data that are expected to be collected by the failed monitors. With the scheme of chunks, the impact of data loss depends on the size of chunks: the larger the chunks, larger the loss. Since collected data are handled in a unit of a chunk and committed to the shared database after a monitor finishes working on the chunk, the failure of a monitor causes the loss of the collected data contained in a working chunk. Such data loss can be avoided by making chunks smaller, however, this will increase the burdens on monitors due to more frequent interaction with the shared database. Therefore, the chunk size is an important factor to decide the robustness of DTS.

In order to investigate the relationship between the chunk size and the interaction with the shared database, we first performed an experiment that invokes the handle of various sizes of chunks. We randomly chose 16 PlanetLab nodes and deployed DTS on them. These nodes are set to the core agents that form a DHT-based database. We also prepared a probing target list that contains 1024 valid IPv4 addresses, and evenly divided them into C chunks $\left(C = 2^i; i = 1, 2, ..., 9\right)$, i.e., each chunk contains $1024/C$ IP addresses. These chunks were stored in the PT DHT. In respective cases, we made all monitors retrieve all chunks from the PT DHT and recorded the monitors behavior.

Fig. 3 illustrates the distribution of required time among all monitors for retrieving one chunk in the respective cases of C. In this figure, the bottom and top of a box respectively show the 25^{th} and 75^{th} percentiles of the required time, and a bold line across a box shows the median value. The ends of a whisker indicate the minimum and maximum values except for the outliers that lie more than 1.5 times IQR (inter-quartile range) lower than the 25^{th} percentile or 1.5 times IQR higher than the 75^{th} percentile. One can see that the required time decreases as the chunk number increases from 1 to 4, however the time just shows a slight change from $C = 4$ to $C = 512$. This is because a dominant element in the required time switches between the chunk size and the overhead caused by the interaction with the PT DHT. In DTS, chunks are exchanged based on the N-TAP's messaging protocol. An N-TAP message usually contains a 16-byte length header, a 47-byte length additional header, and user data. The message is transmitted by TCP. The length of user data increases by 10 bytes per one target IPv4 address. Therefore, the length of the received message for retrieving one chunk is $(63 + 10240/C)$ bytes. Up to $C = 4$, when the message length was 2,623 bytes or more, the dominant part of the required time was the time for transferring a considerable length of a message that contains a chunk. When the value of C was larger than 4, the message length became short enough, meanwhile, the overhead that derives from a routing procedure in DHT cannot be ignored compared to the time for transferring a message.

Then how the chunk size affects the overall workload in the case of the failure of monitors? This is also a considerable problem because DTS ensures monitors' arbitrary joining and leaving and must also be robust to unexpected events, such as monitor failure. In order to deal with this problem, we conducted an experiment that involve the failure of some monitors in process of probing.

For the experiment, we randomly selected 16 PlanetLab nodes that reside in different sites, and deployed DTS on these nodes. We also selected other 16 PlanetLab nodes as probing targets. Then we made the monitors perform the procedures for topology discovery to these targets. We prepared three sizes for chunks: one chunk, 4 chunks, and 16 chunks for 16 targets (these chunks contain the same number of targets without overlapping). Some of the monitors were configured to fail and unexpectedly leave the system at one of these timings: when a monitor performed no probe (0 %), or when a monitor completed probes for 25 %, 50 %, or 75 % of chunks. For example, the proportion of 25 % in the case of 4 chunks means that a monitor fails after it finishes topology probing for one of 4 chunks. We also changed the number of failed monitors between 0, 5, 10, and 15, where the value of 0 means that all monitors finished probing without failure. After the rest of the monitors, i.e., alive monitors, have finished topology discovery to the targets, we looked into the number of probes performed by the alive monitors.

Fig. 4 indicates the number of probes performed on each condition. The number of probes shown in this figure is the average values of the probes performed by alive monitors. From these values, we can find how the failure of monitors on each condition affects the overall workload in DTS. We note that, in the case that the number of chunks is 1, the plots when the proportion of completed chunks is 25 %, 50 %, or 75 % are not shown, because the monitors have only one chunk to handle.

One significant point is that, when monitors have just one chunk, the number of probes scarcely changes depending the number of failed monitors. In this case, the failure of a monitor causes the complete loss of the data collected by the monitor because the data are not committed to the shared database until the monitor finishes the work for only one chunk.

We can also see that, in the case of the number of failure monitors is 15 and the proportion of completed chunks is 0 %, the number of probes shows little change against the variation of the number of chunks. This is because only one monitor kept alive and other monitors failed without performing probes, the alive monitor cannot take advantage of the global stop sets originated by other monitors. As a result, the merit of the Doubletree algorithm is decreased, and the efficiency of topology discovery by the alive monitor was not improved so much. Except for the cases that the number of failure monitors is 15, the number of probes decreases as the chunk size becomes smaller (i. e., the number of chunks gets larger). This means that the smaller chunk size ensures more rapid reflection to the global stop sets, which results in the utilization of the stop sets from other monitors.

Additionally, even if monitors fail, the chunks the failed monitors have already completed contribute to the overall efficiency of topology discovery. As seen in this figure, the higher proportion of completed chunks basically decreases the number of probes more. Especially in the cases where the proportion of completed chunks is 50 % or more, its impact is notable. The reason why we see it brings a bigger impact on the number of probes when the number of chunks is larger (16) will be similar to the one stated in the previous paragraph, i. e., more rapid reflection to the global stop sets.

4 Conclusion

Current systems for discovering the Internet topology at the IP interface level are undergoing a radical shift. Whereas the present generation of systems operates on largely dedicated hosts, numbering between 20 and 200, a new generation of easily downloadable measurement software means that infrastructures based on thousands of hosts could spring up literally overnight. These systems must be carefully engineered in order to avoid abuse and duplication of efforts between tracing monitors. To this end, monitors must share information to guide probing. We stated, in this paper, that this sharing must be decentralized in order to be, among others, scalable and robust. We identified the needs of such a system and discuss how we implement them into what is, to the best of our knowledge, the first fully decentralized tracing system. We are currently exploring the possibilities of our implementation through the investigation of basic characteristics of DTS deployed on the PlanetLab testbed.

References

1. B. Donnet and T. Friedman. Internet topology discovery: a survey. *IEEE Communications Surveys and Tutorials*, 9(4):2–15, December 2007.
2. kc claffy, Y. Hyun, K. Keys, and M. Fomenkov. Internet mapping: from art to science. In *Proc. IEEE CATCH*, March 2009.
3. B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient algorithms for large-scale topology discovery. In *Proc. ACM SIGMETRICS*, June 2005.
4. N. Spring, D. Wetherall, and T. Anderson. Reverse-engineering the internet. In *Proc. HotNets-II*, November 2003.
5. kc claffy, M. Crovella, T. Friedman, C. Shannon, and N. Spring. Community-oriented network measurement infrastructure (COMNI) workshop report. *ACM SIGCOMM Computer Communication Review*, 36(2):41–48, April 2006.
6. Y. Shavitt and E. Shir. DIMES: Let the internet measure itself. *ACM SIGCOMM Computer Communication Review*, 35(5), October 2005.
7. N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, August 2002.
8. N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public internet measurement facility. In *Proc. USENIX USITS*, March 2002.
9. H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An information plance for distributed services. In *Proc. USENIX OSDI*, November 2006.
10. Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. A case study in building layered DHT applications. In *Proc. ACM SIGCOMM*, August 2005.
11. K. Masui and Y. Kadobayashi. N-TAP: A platform of large-scale distributed measurement for overlay network applications. In *Proc. DAS-P2P*, January 2007.
12. I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *IEEE Transactions on Networking (ToN)*, 11(1):17–32, February 2003.
13. K. Masui and Y. Kadobayashi. A role-based peer-to-peer approach to application-oriented measurement platforms. In *Proc. AINTEC*, November 2007.