NAIST-IS-DD0661022

Doctoral Dissertation

Enhancing End-System Capabilities on the Internet with a Large-Scale Observational Approach

Kenji Masui

January 29, 2009

NAIST

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Kenji Masui

Enhancing End-System Capabilities on the Internet with a Large-Scale Observational Approach*

Kenji Masui

Abstract

This dissertation focuses on a new paradigm of network measurement, called application-oriented measurement, which intends the enhancement of end-system capabilities by collecting and providing network characteristics information on the Internet. The Internet provides an easy-to-use pipe for the communication between end systems, concealing the complicated structures and characteristics of its internal components. Such a design principle, so to say that dumb networks simply connect intelligent end systems, has proven its success in the scalable deployment of the Internet, and should be kept in the future. However, as modern network applications running on end systems (e.g., peer-to-peer applications) involve more nodes and networks, they come to realize that the information on network characteristics is indispensable for sustaining and scaling up their services according to the dynamic state of the involved entities. Since the current Internet architecture does not have a mechanism for explicitly providing such information to end systems, these applications have been experiencing difficulties in grasping the state of the Internet. For supporting more flexible activities of network applications with keeping the design principle above, we choose an approach of the large-scale Internet observation for collecting network characteristics information for the applications.

The ultimate goal of our study is the actual deployment of a network service of application-oriented measurement. Towards this goal, we mainly contribute in the following three topics: measurement platform, measurement methodology, and its application. We first explore the fundamental requirements of application-oriented measurement with referring the kinds of network characteristics required by existing and future large-scale applications. Considering these requirements, we design and implement an application-oriented measurement platform (AOMP), which forms an overlay network for measurement activities, as a case study. We validate its feasibility by the performance evaluation on actual networks and the demonstration of an application that utilizes our AOMP. Additionally, we implement an algorithm for the network topology discovery in a cooperative manner named Doubletree on this platform, and investigate its advantages for the large-scale Internet observation. Through these research and experiments, we finally lead a comprehensive discussion about our contribution, related work, and open issues for the ultimate goal.

Keywords:

Internet observation, network measurement, peer-to-peer network, application-oriented measurement platform, cooperative measurement methodology.

*Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0661022, January 29, 2009.

Acknowledgements

This dissertation is a milestone for my research project on the Internet observation. Since its launch at Nara Institute of Science and Technology (NAIST) in 2005, a lot of people have kindly given help, support and guidance in my doctoral work, and I could never have reached the current stage without them. I would like to acknowledge these people here.

First, I wish to express my gratitude to my mentor, Dr. Suguru Yamaguchi, for his energetic encouragement, depthful insights, and generosity. His presence really encouraged me to be engaged in the research of the Internet and networking technology. I also wish to express my gratitude to another mentor, Dr. Youki Kadobayashi, for his excellent guidance, providing me with research opportunities and environments, and showing his researcher's spirit. I learned much from him in logical thinking, scientific writing, being confident in my work, and so on. I would like to thank my committee members, Dr. Hideki Sunahara and Dr. Hiroshi Esaki, for their helpful support in the improvement of this dissertation. Their comments and suggestions brought a broader point of view on me.

I warmly thank the faculty members of Internet Engineering Laboratory (IPLab) at NAIST: Mr. Takeshi Okuda, Dr. Shigeru Kashihara, and Dr. Hiroaki Hazeyama, for their guidance on my research activities. I would like to thank the past and present IPLab members, with whom I spent a stimulating laboratory life. The precious experience for four and a quarter years at IPLab is unforgettable and cannot be gained elsewhere.

During my stay in Paris in 2006 as an exchange student in the CNRS-WIDE collaboration project, Dr. Timur Friedman at Laboratoire d'informatique de Paris 6 (LIP6), Université Pierre et Marie Curie (UPMC) in France, and Dr. Kavé Salamatian at Lancaster University in the United Kingdom (he was formerly a member of LIP6, too), supported both my research activities and daily life in Paris. I wish to thank these two mentors, and other students and staffs who met through this project as well. Moreover, I would like to send a note of gratitude to Dr. Kenjiro Cho at Internet Initiative Japan (IIJ) for arranging the student exchange program as a leader of the measurement team in the project.

I am truly indebted to Dr. Olivier Bonaventure and Dr. Benoit Donnet, who kindly invited me to a one-month research collaboration with their research group, IP Networking Lab at Université catholique de Louvain (UCL) in Belgium. I would also like to thank other members for fruitful collaboration and discussion.

A part of my research activities was supported by the following programs: Strategic International Cooperative Program by Japan Science and Technology Agency (JST), Creative and International Competitiveness Project (CICP) 2007 by NAIST, and Research Fellowship for Young Scientists by Japan Society for the Promotion of Science (JSPS). Thanks to the parties concerned in the respective programs.

I cannot conclude this dissertation without my appreciative words for the chefs and staffs of

Tentenyu, a ramen restaurant located at Ichijoji, Kyoto. Their ramen has been a building block of me for nearly a decade, of course longer than the period of my research activities. It has been a place of relaxing and recharging my batteries. I only regret that they don't yet have any restaurant in Tokyo.

Finally, my very special thanks to my family. I'm here now, thanks to you.

January 29, 2009

Contents

List of Figures

List of Tables

Chapter 1

Introduction

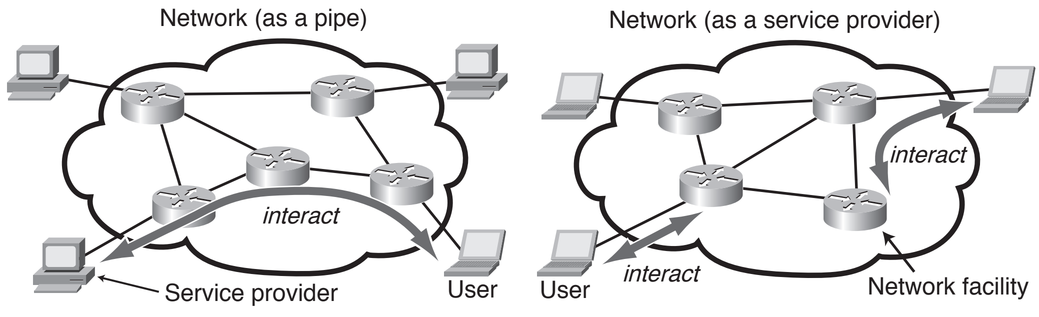
The Internet has evolved together with a ton of appealing network applications and services. Since its first appearance in the 1980s, the Internet has continued to become larger and more casual. In the beginning, it was a communication network only for the particular groups of people like academia and military personnel, however, today it provides universal communication for people throughout the world. After the success of the popularization of the Internet in the 1990s, we have seen lots of interesting Internet-based services and communication among people on the services, even though it was an event in a few decades. Electronic mail (email) and content distribution on World Wide Web (WWW) are traditional but still fundamental applications for most people. Additionally, the recent improvement of network environments has brought the rise of new services such as media streaming, peer-to-peer file sharing and multi-person audio/video conference. These applications, which make the Internet vivid, energetic, and amusing, basically provide their services by the interaction between end systems, on which users reside. The Internet is just a pipe without the applications.

Our study described in this dissertation supports such activities by the end systems with an approach of large-scale network measurement. What if existing applications get a hint for better performance and stability? Or what if a new application is struggling with the realization of its innovative service on the Internet? Our answer is *"knowledge is power."* Letting the applications know more about the state of the Internet, we expect the improvement of existing applications and the emergence of novel applications. And one of the ways for acquiring such knowledge is the network measurement at the Internet scale. We focus on how to collect and provide the knowledge on the Internet for enhancing the capabilities of applications running on end systems. Put simply, we believe that an observation satellite for the Internet can bring the change in the potential of the Internet.

1.1 The Internet and the End-to-End Principle

Why has the Internet succeeded in its large-scale deployment? One of the reasons lies in its design principle. Before talking about the current trend of large-scale network applications, we start with referring to the design of the underlying network.

For the design of computer network, there exist two contrasting models; one is dumb network and the other is intelligent network, as respectively depicted in Figures 1.1(a) and 1.1(b). The word "intelligence" here means a capability of processing network traffic and controlling network structure more than just transmitting traffic from point to point. In the model of dumb network,



(a) Dumb network (b) Intelligent network

Figure 1.1: Designs of computer network.

a network itself has little intelligence and is kept simple. Therefore, the network dedicates itself to the transmission of network traffic, and basically does nothing other than that. From the end system's view, the dumb network looks like a pipe that just carries user traffic as they desire, and two end systems are directly connected through the network and can interact between them, as shown in the figure. They also have to manage to coordinate among other end systems explicitly or implicitly in case of sharing the network resource (and they may even have a power to corrupt the network). On the other hand, in the model of intelligent network, a network does more than that if needed; providing application services, managing resource allocation voluntarily, establishing a special route for the traffic from prioritized customer, blocking bad users automatically, and so on. Under the centralized management by the intelligent network, a burden on end systems is generally lighter than that in the case of the dumb network; because some tasks such as resource allocation can be handled within the intelligent network. In the complete intelligent network, the network works as a service provider or a gateway for destination, and makes the coordination of all components in the network, hence, an end system, which is basically dumb, does not directly interact with other end systems but interact via the network, as shown in the figure.

The Internet, which is one of the operational computer networks, was designed based on the "end-to-end principle [1]." This principle states that a communication system and its protocols should be designed as the processes involved in communication are done at end systems rather than intermediate components as possible. The idea can be clearly found in the Internet protocol suite, or TCP/IP. IP (Internet Protocol) [2] is a protocol which essentially provides a feature of the point-to-point transmission of network traffic, or packets in other words. Meanwhile, TCP (Transmission Control Protocol) [3] has more intelligent features such like error detection, retransmission, and flow control. In the Internet, networks handle only the IP, and TCP and other higher-layer protocols are implemented only on end systems; i.e., the Internet assumes that the networks are dumb and the end systems are intelligent. As just described, the choice of the Internet is the model of dumb network, naturally resulting from its design principle.

Eventually, the end-to-end principle contributed in the large-scale deployment of the Internet. The simplicity of networks and the decentralization of intelligence among end systems have proven that this principle can apply to a design of an operational network. Moreover, the Internet looks simple to end systems, because it provides an easy-to-use and operational function of packet transmission. End systems do not have to care what and how the internal components of the Internet are, but just have to throw their packets to the network. Such a convenient pipe is attractive enough for end systems and they can concentrate on constructing their services.

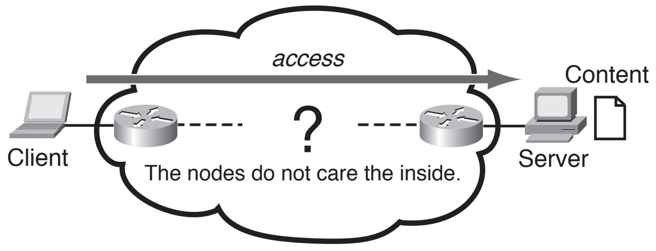
Actually, the current Internet adopts a hybrid model of the dumb and intelligent network models. For example, some ISPs (Internet Service Providers) will restrict the available bandwidth for the flows from some applications by identifying them from user traffic. Or we can see a lot of firewalls and load balancers, which inspect the headers of transmission protocols and payloads, on actual networks. However, we can say that the Internet has continued to grow prioritizing the end-to-end principle, because these intermediate systems usually perform additional processes for packet transmission, and an end system still does the interaction among other end systems as a service provider or a consumer. On the other hand, we come to see that some researchers have begun to explore an architecture of new generation network [4, 5]. They sometimes mention that we should once discard the current design of the Internet and reconstruct it with a clean-slate approach or an intelligent network approach. However, this trend cannot be immediately a negative stuff against the end-to-end principle, because there does not yet exist another design principle that excels the current one in terms of the actual deployment. The end-to-end principle, whose success was proven by the history of the Internet, will be worth of being kept in the future Internet.

1.2 The Rise of Autonomic Network Applications

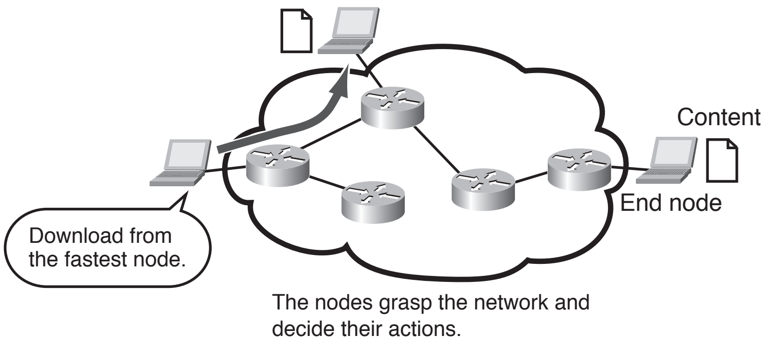
As the Internet grows larger, applications running on end systems have also become more various in their forms and purposes. Especially, one of the big transitions is the rise of autonomic network applications. Reviewing the transition from traditional applications to autonomic ones, we summarize the autonomy of network applications in this section.

Traditional network applications, such as email and WWW, work under the rules of the clear division of roles and the disinterest towards networks. This matter is depicted in Figure 1.2(a). The first rule means that the infrastructure for the application's service is relatively fixed and the roles of involved nodes are preliminarily defined. This manner is often referred to as the client-server model. In the case of WWW, an HTTP (Hypertext Transfer Protocol) [6] server is usually placed at the side of a service provider, and a user behaves as a client by accessing to the server from start to end. The second rule means that servers and clients do not care how the intermediate network carries their packets, as presented in the explanation of dumb network in Section 1.1. Actually these applications do not know the structure or characteristics of the internal components of the Internet like nodes and links, but try to send their application messages only by specifying destinations. This is the most simple manner of utilizing the Internet as a universal communication platform.

On the other hand, we have seen the rise of the network applications that involve more complicated autonomic procedures for several years [7]. An autonomic network application decides its future behavior according to the structure and status of networks. We show some cases that indicate the autonomy of network applications. For the content distribution on the Internet, if multiple nodes (suppliers) have same content wanted by a node, the node can select a supplier based on the measurement result of the bandwidth between the node and suppliers so as to finish downloading the content faster (shown in Figure 1.2(b)). Or if the application finds that a path



(a) Traditional client-server application


Content
End node
Download from the fastest node.
The nodes grasp the network and decide their actions.

(b) Autonomic application

Figure 1.2: Application’s autonomy.

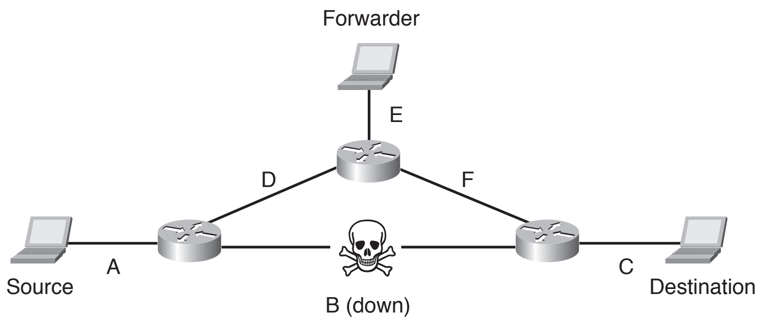


Figure 1.3: Recovery from path failure with the aid of a data forwarder.

from itself to a destination node is not working well for a reason that a part of the path is broken, it will try to send a message to other node and ask it to forward the message to the destination node. If the paths from the source to the forwarder and from the forwarder to the destination do not contain the broken path, the message will be sent successfully. This behavior can be simply explained with Figure 1.3. The source node in the figure tries to send some packets from itself to the destination, and the packets traverse the links A, B and C on the IP network. When the link B fails, the source will sense the failure by some anomalies like the drops of TCP ACK packets from the destination, and try to find an alternative path for delivering packets to the destination node. If the source node has a cooperative node that forwards received packets to another node (usually called a forwarder) and the source knows the IP topology among the source, destination and forwarder nodes, the source can find whether its packets can be delivered with help from the forwarder. In the case of the figure, if the path from the source to the forwarder is A – D – E and the path from the forwarder to the destination is E – F – C, the packets from the source can be delivered via the forwarder even if the link B is down because such a path, i.e., A – D – E – F – C, does not contain the link B. Even though the IP network itself often tries to recover from the path failure with IP dynamic routing protocols such as OSPF [8], such a procedure usually takes a considerable amount of time for the convergence of IP network paths (usually from several minutes to a few hours). The autonomic applications do not need to wait this convergence but can immediately take an action with the knowledge of underlying networks. Another example of autonomic behaviors is These actions are often seen in the research of large-scale peer-to-peer network applications, with which end nodes aggressively interact each other.

These applications do not necessarily have the same characteristics as the traditional ones stated above, because the autonomic applications have dynamism on their behavior rather than that the traditional ones work statically. For example, in peer-to-peer network applications, a node can be both a client and a server, and its role dynamically changes. In such a situation, the infrastructure for these services cannot be fixed and end users and their systems can also be a part of the infrastructure. Additionally, the Internet is not just a pipe for these applications. For the autonomic actions described above, the applications have to know the status of the Internet, such as available bandwidth and link failure. That is, beyond the naive utilization of the Internet, these applications behave in sophisticated ways for more attractive and complicated purposes.

Here we define the term "network characteristics" for further discussion. Network characteristics indicate the information that characterizes network elements, such as a node and a link. The typical instances of the network characteristics information include IP-level network topology, round-trip time (RTT) of the communication between two nodes, and the number of an AS to which a node belongs. As stated in Section 1.2, the network characteristics information is utilized by the autonomic applications as the criteria for their behavior. The detail of network characteristics will be described in Section 2.1.

In the future, more autonomic applications will appear on the Internet and they will serve attractive and innovative services that cannot be realized by existing applications. For supporting these services, the Internet will be expected to be more powerful in its performance and have a supportive service for the applications.

1.3 A Boundary of End-System Capabilities on the Current Internet

Even though the network characteristics information is indispensable for the autonomic applications as stated in Section 1.2, these applications have been experiencing difficulties in obtaining such information. This situation can be understood from the fact that a lot of researchers are making efforts in the field of the application's autonomy [9, 10, 11] by the theoretical verification and simulation experiments but these results are seldom implemented on actual applications. More precisely, existing autonomic applications sometimes performs the simple measurement of the network characteristics information such as RTT, and utilize it for the decisions of their behaviors, but there is disjunction between the researches and the applications. The researches on the autonomy of applications often insists that the applications can be improved given that the target network characteristics information can be obtained, that is, how to obtain these information is left. The applications do not know how to collect the information, or know how difficult the collection is, so the research results are hardly reflected to the applications.

This is because the Internet itself does not provide a mechanism which explicitly informs end systems of such information. When an end system communicates on the Internet, it usually uses a simplified interface like a network socket, which is implemented on most of the modern operating systems. The socket is easy-to-use because an end system can communicate with other systems only by specifying destination addresses, however, such a simplified interface conceals the structure and characteristics of the internal components of the Internet, or the network characteristics information, at the same time. For instance, in case where an application running on an end system establishes a TCP connection between another node with a socket, what the application has to know is only the destination IP address and port number. The application does not (and basically cannot) know the complicated structure of the overall Internet, or even does not know what kind of intermediate nodes there are on the path.

Under such a trade-off between simplicity and informativeness, the most common interface adopts the side of simplicity. According to the design principle of the Internet, the current situation is a natural result. Networking, or transmitting packets is an explicit role for networks, not end systems, on the Internet. Under this circumstance, a socket is the simplest interface that works with a few kinds of identifiers, i.e., a node's address and a port number. Its minimum functionality is good enough from a perspective of functional partition in system design, and has surely supported the deployment of traditional applications.

However, considering the continuous growth of the Internet, large-scale network services provided by autonomic applications will become more vital than ever, and a mechanism that supports these services should be prepared. If the Internet continues to ignore the requirements from these applications, the applications on end systems may stop their evolution due to that the ideas in a phase of research are not actually leveraged. Given that the services by the applications have made the Internet attractive for a lot of people, we still believe that the Internet platform should be improved so as to let network applications behave without bonds.

In summary, what the current Internet lacks is a function of exporting its status. Due to this, end systems and autonomic applications running on them has been experiencing the difficulty of the operation responding to the state of the Internet, or the network characteristics information. Currently, for obtaining this information, an additional procedure such as network measurement should be done by some entities. For a lacking piece, now it's time to seek an answer.

1.4 How Should We Support the Autonomic Applications?

Because of the constraints presented in Section 1.3, we have not yet witnessed the behavior of responding to the dynamic state of the Internet in large-scale network applications. Then how should we support this kind of behavior? In this section, we argue on this subject.

First, even if we have to modify the current architecture of the Internet or add a new component for supporting the autonomic applications, we believe that the end-to-end principle should be still kept as referred to in Section 1.2. The model of intelligent systems interconnected on dumb networks is reasonable for better scalability of a network system. Since the most of autonomic applications assume that they work with involving a number of end systems residing in multiple administrative domains, their scalability is an important factor to be considered. Therefore, the expansion should be done without destroying the idea of the end-to-end principle. Moreover, and we have a lot of existing technologies and facilities designed for this model in the fields of application, performance, security, and so on. Keeping this model will have an advantage in leveraging these resources.

Secondly, the support should not expect the unrealistic capability of end systems. Of course, once forgetting existing boundaries is often important for researches to produce a novel idea. However, we are now facing the problem of the boundary of the actual system, therefore, such the expectation may be so rash. At the same time, no one can assure that the same kind of the problem still occurs in such the imaginary world. Learning what the current Internet can do and cannot do, and accordingly planning a strategy for solving the problem are important. In short, evolutionary progress is preferred for the contribution to actual systems.

Thirdly, the support should be independent and easy-to-use for the autonomic applications. This thought comes from a lesson from most of the existing Internet protocols and applications. For example, well-deployed services such as email, WWW, and Domain Name System (DNS) do not interfere in the internal state of other services, and they keep to the utilization of public interfaces of other services. In addition, though these applications handle complicated procedures in their internals, they just provide simple input/output interfaces, which motivate users in using their services.

Considering these thoughts, our choice is a service of application-oriented network measurement for end systems, which involves the large-scale Internet observation. By observing the Internet in large-scale, we can expect that the network characteristics information required by autonomic applications will be collected and the service can provide it for the applications. Though the procedures of the large-scale network measurement are generally complicated, abstracting them into a service will make them easy-to-use for users, or autonomic applications. Moreover, the manner of an independent network service will not be against the end-to-end principle if the service itself is composed of end systems.

However, unlike the measurement performed in traditional measurement platforms, application-oriented measurement has different requirements for its goal. For example, the traditional measurement takes mid-term or long-term period (e.g., from a few hours to several years) for collecting the network characteristics information, however, the autonomic applications cannot wait for such a long time because they have to make a judgment about their future behaviors as soon as possible. Considering the gap between traditional measurement and application-oriented measurement, we have to carefully design a service platform for application-oriented measurement. These considerations will be described in Chapter 2.

Consequently, our approach for supporting the autonomic applications, which results in the enhancement of end-system capabilities, is the large-scale Internet observation. For the better environment for applications, we study on a large-scale network measurement service, considering its actual deployment, believing that the knowledge acquired by measurement brings power on the applications.

1.5 Positions and Contributions

The ultimate goal of our study is the actual deployment of a network service of application-oriented measurement. Before the deployment, we need to design this totally novel system carefully. For this process, we consider the following rules according to the arguments so far.

- For supporting the autonomic applications, we adopt an approach of a network service of the Internet observation, or the large-scale network measurement for enhancing end-system capabilities.
- The design of our network measurement service emphasizes the current architecture of the Internet and the end-to-end principle.
- Our network measurement service does not disrupt existing services on the Internet. The service concentrates on collecting and providing the network characteristics information.

Towards the ultimate goal, in this dissertation, we make the contributions below which will be steps forward for the improvement of the current situation:

- We explore the requirements of application-oriented network measurement, and formulate a guideline for its deployment on the actual network environment. Since the requirements for the measurement for autonomic applications are yet to be clearly defined, this will be a basis for the future research in this field.
- Based on the defined requirements, we propose a measurement network for a service of application-oriented network measurement. With the implementation of our application-oriented measurement platform which utilizes this measurement network, we evaluate the capability and performance of this platform on the actual network, and we discuss the feasibility of a network measurement service built upon our proposed measurement network.
- On our platform, we actually implement one of the large-scale measurement methodologies, and investigate its behavior on the actual network. Given that most of such kind of the measurement methodologies are only validated with simulation experiments, our work will show findings for their actual deployment.
- Through these investigations and experiments, we extract realistic open issues for the actual deployment of application-oriented network measurement services from a macroscopic view.

We agree that our approach is not one and only answer to the problem described above. We also agree that our study does not provide a perfect answer to the ultimate goal, or the actual deployment, which is difficult to be evaluated with specific indices. However, we summarize our research contributions in this dissertation in order to take a step forward, for a new paradigm of network measurement and autonomic applications.

1.6 Organization of this Dissertation

In this chapter, we have described the nature of the Internet, its merits, problems caused by the nature, and our motivation and position for improving the current Internet. The rest of this dissertation continues as follows. Chapter 2 investigates what kind of network characteristics information is needed by autonomic applications and then defines the fundamental requirements of our approach, the application-oriented network measurement, based on the investigation. In Chapter 3, we introduce the design and implementation of an application-oriented measurement platform called N-TAP, which leverages a peer-to-peer networking technique for deploying a large-scale measurement network. We evaluate its capability and performance in the actual network environment, and validate whether N-TAP can be deployed as an actual network service according to the requirements presented in the previous chapter. Next, Chapter 4 is a part for the field work of large-scale application-oriented measurement with the Doubletree algorithm [12]. We investigate the basic characteristics and behavior of the Doubletree algorithm, which was not yet tested on the actual network. Chapter 5 follows with a comprehensive discussion towards the our ultimate goal. We review the fundamental requirements for application-oriented network measurement and discuss open issues and future work. And finally, Chapter 6 concludes this thesis in point of our contributions and the future direction of the application-oriented network measurement.

Chapter 2

Application-Oriented Measurement Platform

Application-oriented network measurement is a new trend of network measurement. For its goal, i.e., the provision of the network characteristics information to applications, we will see its new and different requirements, compared to the ones of the traditional network measurement. On its scalability, responsiveness, measurement accuracy and so on, the requirements are more severe and the procedures of the measurement are also complicated. Due to that these requirements are not yet explicitly defined, the preliminary analysis of these requirements is significant for future study.

In this chapter, we start with the definitions of network characteristics and Internet observation for further discussions. Then, consulting the requirements, we design a platform on which application-oriented measurement is performed. We call such a platform an application-oriented measurement platform, or an AOMP in short. AOMP is an infrastructure for a service of application-oriented network measurement. Through these desk studies, we prepare the implementation of an actual AOMP system.

2.1 Network Characteristics Information

The term "network characteristics information" generally means the information that characterizes the entities on the Internet. The entities include a node, a link, a group of them, and the communication between nodes. In other words, the network characteristics information indicates the state of the entities on the Internet. As stated in Section 1.2, the network characteristics information is indispensable for autonomic applications because they utilize this kind of information for making decisions on their future behavior.

We have a various instances of the network characteristics information. Here we categorize them into several groups as shown in Table 2.1 and explain them respectively. In this table, we present the types of the network characteristics information, their descriptions, and their instances. The entities in brackets are the targets of respective instances, and an instance give the information on its target. The categories are the following five: identifier, specification, configuration, performance, and structure. The identifier provides the information that distinguishes one entity among others of the same sort. Generally, the identifier is unique within a specific area so as to distinguish an entity correctly. The identifier includes a MAC address for a network

Table 2.1: Instances of network characteristic information.

Type	Description and examples	
Identifier	Information that recognizes an entity correctly among entities of the same sort	
	Examples:	MAC address [network interface]
		IP address [device (node)]
		Port number [service on a node]
		AS number [a group of nodes and links in the same administrative domain]
Specification	Information that indicates a predefined function utilized for a specific purpose	
	Examples:	Media type [link]
		Protocol [communication between multiple nodes]
Configuration	Information given to run a function	
	Examples:	MTU [link]
		OSPF cost [link]
		TCP receive buffer size [node]
Performance	Quality index of a given function	
	Examples:	Packet loss rate [communication path]
		Round-trip time (RTT) [communication path]
		Available bandwidth [communication path]
		Service uptime [service]
Structure	Information of the relationship among multiple entities	
	Examples:	IP hop count [two nodes]
		IP topology [multiple nodes]
		AS topology [multiple ASes]

interface card (NIC), an IP address for a node in a network, and so on. The second one is specification, which shows what kind of functions is used for providing a target feature. The specifications of these functions are usually predefined. One instance of the specification information is protocol, for example, TCP or UDP for realizing the transport feature. The third is the configuration information, or the parameters for the specification. The configuration information cannot exist by itself, but can with some predefined functions, or specifications. This information determines the detailed behavior of the procedures in the specification. For example, OSPF (Open Shortest Path First) [8] cost values are given to the links managed with the OSPF protocol in order to determine the preference of its utilization for transferring packets. The fourth is the performance information. This information can be regarded as the indices of the quality of provided functions. Packet loss rate is often used to indicate whether a target link or path is working correctly or not, and RTT is also used in the similar situation. The last one is the structure information, which presents the relationships among multiple entities. IP topology, one of this kind of the network characteristics, is the data in graph form, which describes how the nodes identified by IP addresses are directly connected for packet transmission. As seen above, we have various kinds of network characteristics information, and they are different in their roles.

On the Internet, the network characteristics information dynamically changes over time. One of its reasons is the fact that the Internet is naturally a distributed system in point of administration. No one can manage the entire Internet but everyone can manage some nodes and links within at least his or her administrative domain, e.g., from a personal device to an enterprise network. Therefore, we can change the identifier, specification, configuration, or structure information anytime and this may bring the change of the performance information. Additionally, since the Internet is a best-effort service, its quality tends to fluctuate, and this brings the changes on the performance and structure information. The Internet is a quite large system, hence the continuous changes are produced as the results of the activities of a great number of entities.

2.2 Internet Observation

Internet observation is the process of discovering, investigating, and revealing the structure and characteristics of the components of the Internet such as nodes and links, and its coverage is basically the entire Internet. Thinking of the arguments so far, the Internet observation can be regarded as a procedure of collecting network characteristics information. In this section, we first introduce three methodologies of the Internet observation.

The first one is the access to the information itself. This methodology acquires the target data with accessing a storage where the data is contained or using some APIs (Application Program Interfaces). The storage can be an area of memory, a configuration file in a disk, and so on, and the APIs for fetching these data are often provided, too. In this case, the target data is preliminarily defined and we fetch it with some methods. Therefore, this methodology is often used for obtaining the identifier, specification, and configuration information of a local system. Or sometimes it involves an access through a network such as obtaining a node's status with SNMP (Simple Network Management Protocol) [13].

The rest of the methodologies are the passive measurement and active measurement, which are often called network measurement collectively. The procedures of passive measurement basically do not involve the produce or modification of network traffic, but try to obtain the target data just with watching the target's state. A simple example of the passive measurement is the network traffic monitoring. With the monitoring, we can obtain the information such as the amount of received/sent traffic. The active measurement, in contrast, produces some traffic for obtaining the target data. As one methodology of the active measurement, we often use the technique of ping. By leveraging the echo request/response features of the ICMP [14] protocol, it can find the required time for the packet's both-way travel between the nodes, and packet loss rate on the path, too.

After collecting the target information, the Internet observation sometimes involves a procedure of statistical calculation. This procedure occurs if the target data is calculated from the collected data. We use various algorithms for the computation, and their complexity also varies respectively. One of the simple examples is the calculation of the moving average of RTT. In this case, we continuously collect the RTT data between two nodes, and at a certain point, we calculate the mean value of the recent values of measured RTT.

Here we note that all of these observation methodologies have impacts on the network characteristics information, whether the information is the target one or not. This can be explained by a natural law, which insists that a procedure of observation brings more or less changes in a system. The degree of the impacts are generally less in the case of the passive measurement compared to the case of the active measurement, because the passive measurement does not bring any touch in a network system and the active measurement does. However, the degree possibly changes in respective cases and we cannot present a grand theory for its order. Giving an extreme case, what if a methodology of the passive measurement requires the quite frequent access to the local disk and the calculation that spends 24 hours even though it does not involve any network access? It may slow down or panic the computer, and this will be more serious change in a system than just performing one-time ping.

Recently, there is a trend of more sophisticated measurement methodologies, called cooperative measurement. Commonly, the procedures of traditional measurement methodologies are completed within the proactive activities by one node. For example, in the technique of ping, only one source node has a motivation to measure RTT, and it throws a ping packets into a network. Though a destination node receives these packets and respond with pong packets, it does neither know why the source node threw the packets nor have the same motivation. On the other, the cooperative measurement involves two or more nodes and they interact with each other according to a specific algorithm for the achievement of the same goal. The cooperative measurement methodologies are different from the above three traditional methodologies in point of involving multiple proactive nodes, and the procedures of the cooperative ones are sometimes composed of one or more of the traditional ones.

With the cooperative measurement, we can obtain more various kinds of the network characteristics in a larger scale, owing to the involvement of multiple proactive nodes. The simplest cooperative measurement is to ask other nodes for collecting the target data. Though one monitoring node cannot know RTT from another source node to others by itself with the technique of ping, it can know if it can ask the source node to measure the target RTT and provide the measured data. Moreover, the cooperation among multiple nodes and estimation algorithms bring a power of large-scale network measurement. Let us say, the Vivaldi algorithm [15] is one of the cooperative measurement methodologies. Its goal is to obtain the matrix of full-meshed RTT data for a group of monitoring nodes. Since the number of the matrix elements is n^2, where n denotes the number of the monitoring nodes, measuring RTT data for all elements will be difficult when n becomes larger. In Vivaldi, they measure RTT only for a part of the elements and try to estimate RTT for the rest of the elements. Their key idea is to plot the monitoring nodes in Euclidean space and define the distance on it as RTT between two nodes, and continuously modify their coordinates with the physical mass-spring model and measured RTT data. Each monitoring node communicates with other nodes for exchanging their position data in order to fix node's positions on the coordinate. With their simulation results, they confirmed that the positions of the nodes converge become stable after a period and the distance values can provide estimated RTT with a good accuracy. We can also find a cooperative measurement methodology with the passive measurement of the IP TTL values of arriving packets [16]. This methodology realizes the inference of the macroscopic network topology through the collection of TTL values observed on monitoring nodes. As seen above, the cooperative measurement is expected to be a good solution of the Internet observation, especially in the large-scale one.

2.3 Application-Oriented Network Measurement

So far, we have referred to the network characteristics information and the Internet observation, or the collection of the network characteristics information. In this section, more concretely, we describe application-oriented network measurement and its requirements.

The term "application-oriented network measurement" means the network measurement whose purpose is to collect and provide the network characteristics information for network applications, especially autonomic ones as described in Chapter 1. The application-oriented network measurement is a subset of network measurement from the viewpoint of the classification based on its purpose. Each application has its own objective and the provided information is utilized for achieving the objective. The types of the required network characteristics information depends on these objectives, so we start with investigating them.

2.3.1 Requirements from Autonomic Applications

Here we enumerate several purposes for which autonomic applications utilize network characteristics information. For each case, we analyze the requirements for obtaining such information.

Optimization of Structured Peer-to-Peer Networks

A structured peer-to-peer network locates its participant nodes employing a common rule for efficient routing on the peer-to-peer network. The most popular structured peer-to-peer network is the distributed hash table (DHT), which provides a function of storing and retrieving key/value pairs in a manner of a decentralized network system. DHT-based systems often try to optimize their overlay networks for improving the performance of querying.

In CAN [17, 18], a DHT implementation, they found that the mismatch between the underlying topology (i.e., IP topology) and the overlay topology causes slower querying on CAN, and introduced the topologically-sensitive construction of its overlay network. This methodology first makes each CAN node measures the RTTs to multiple landmark nodes, and finds an order of the RTTs. Based on the order, each CAN node is located in the corresponding zone in the CAN's coordinate space, which results in neighbors in the coordinate space being topologically close on the underlying network. For improving the responsiveness on the Chord [19] network as well, they propose an idea that lets each Chord node have a set of alternate nodes for each finger in addition to ordinary fingers. By measuring the RTTs from the Chord node to the alternate nodes, they calculate the values of network proximity metric for respective alternate nodes. They proved that selecting a successor based on the network proximity metric instead of selecting the largest finger considerably improves the latency of the Chord's overlay querying. Tapestry [20] exploits the RTT and loss-late information among its participant nodes for the optimization as well. Pastry [21] also puts weight on the locality of overlay nodes and adopts the scalar value called distance which determines the locality between the overlay nodes. The Pastry algorithm itself does not specify what kind of information should be chosen for the distance value and let its applications decide, however they showed several metrics such as IP routing hops, IP topology among the overlay nodes. Xu et al. also proposed a methodology for building topology-aware overlay networks based on the RTT measurement among overlay nodes [22].

For the structured peer-to-peer networks, we can summarize the requirements for obtaining network characteristics information as follows:

Procedure Modification of the nodes' location or routing table information based on the locality metrics.

Purpose To improve the latency of overlay querying, which causes the overall performance degradation on their applications.

Required Information

- Type — RTT, IP hop count, IP topology, loss-rate, and other performance metrics.
- Targets — Overlay nodes, or sometimes include special nodes such as landmark nodes.
- Accuracy — Information that reflects the current status. This may involve the collection of the current value of the target information, or the estimation of it from the previously-collected time-series data.
- Time — ASAP. The optimization procedure cannot be started until the measurement finishes.

Construction of Application-Layer Multicast Trees

The technique of the application-layer multicast (ALM) [23] is often used for delivering contents to multiple destinations. The type of the delivered contents changes by the applications, for example, live-streaming videos, large files, and short messages. Depending on the applications' purposes, they construct their own multicast trees on top of the underlying networks, which does not need the support of the multicast function on the layer of the IP network.

Overcast [24] is an ALM application designed for transferring large-size contents. Its multicast tree is rooted at the source and the tree is constructed so as to maximize the available bandwidth between root and other nodes. In particular, the overlay nodes first measure the available bandwidth among them and select faster nodes as their neighbors. If the values of the available bandwidth to two nodes are almost same, then a node probes IP topology with the technique of traceroute and finds more proximate node. Scribe [25] is an event-notification overlay and also measures end-to-end bandwidth for the construction of its multicast tree. On the other hand, Narada [26] is intended for an infrastructure of delay-sensitive applications such as VoIP and live streaming. They measure the end-to-end RTTs among its participant nodes and add or remove links to/from its multicast tree based on the results of the measurement. Liu et al. introduced Adaptive Connection Establishment (ACE), an algorithm that builds an overlay multicast tree among each source node and the peers within a certain diameter from the source peer. They utilize the minimum spanning tree algorithm for the tree construction and the cost on links is calculated from network delay, therefore, the RTT measurement among overlay nodes will be needed in an actual system. Other metrics that have linearity can be adopted as the cost metric in ACE. We can find other studies [27, 28] that use the topology data for optimizing the ALM tree.

For the ALM applications, we can summarize the requirements for obtaining network characteristics information as follows:

Procedure Construction or modification of the application-layer multicast tree based on the proximity metrics.

Purpose To improve a specific performance for delivering contents, such as latency and bandwidth.

Required Information

- Type — RTT, IP hop count, IP topology, available bandwidth, and other performance metrics.
- Targets — Overlay nodes.
- Accuracy — Information that reflects the current status.
- Time — ASAP. The tree construction procedure cannot be started until the measurement finishes.

Proximity-Based Node Selection

Many existing applications try to find a proximate node among candidate nodes so as to increase the performance of the applications. The term "proximate" here means that nodes are proximally located in the space of a specific performance metric such as RTT, available bandwidth, and service uptime. This technique is sometimes a part of the procedures of the overlay network optimization and multicast tree construction stated above, but is different in point of that it does not include further optimization procedures and starts the procedures for application's services just after the selection. Gnutella and some other peer-to-peer file sharing applications select nearby nodes as peering nodes according to the RTT measurement results between the source node and candidate nodes.

Another popular way to find a proximate node is to exploit the node's location by looking up the list of subnets that is preliminarily prepared. For example, the Akamai [29]'s DNS service replies the IP address of a proximate service host among all the service hosts distributed in multiple locations by checking the IP address of a client node and finding its location from the address list. However, the end-to-end measurement has several advantages against such a well-prepared approach. By performing the end-to-end measurement, no one needs to maintain the information such as the address list. Additionally, the end-to-end measurement approach can select more appropriate metrics for improving the application's performance because such metrics directly affect the performance while the address list information can be obsolete and does not directly indicate performance.

For the applications that take an approach of the proximity-based node selection, we can summarize the requirements for obtaining network characteristics information as follows:

Procedure Measurement just before starting the procedures for an application service.

Purpose To improve the performance of an application service.

Required Information

- Type — RTT, available bandwidth, and other performance metrics.
- Targets — Nodes involved in a service.
- Accuracy — Information that reflects the current status.
- Time — ASAP. The service procedure cannot be started until the measurement finishes.

Failure Monitoring

The failure monitoring is an important process for autonomic applications to take a recovery action against the unexpected failure. RON (Resilient Overlay Network) [30], which was introduced by D. Andersen, et al., provides an architecture for constructing an overlay network with the durability against path failures and periods of degraded performance. RON monitors its own virtual links with measuring RTT and packet loss rate of the links as a performance index. In addition, we can suppose the switch of overlay routes in the case of the performance degradation on specific links.

For the application's failure monitoring, we can summarize the requirements for obtaining network characteristics information as follows:

Procedure Finding a failure and taking an action against it.

Purpose To sustain an application service.

Required Information

- Type — RTT, available bandwidth, and other performance metrics.
- Targets — Nodes involved in a service.
- Accuracy — Information that reflects the current status.
- Time — ASAP. The service may stop while being unaware of the failure.

General Discussions

Nakao et al. presented several primitive operations for constructing an efficient and scalable overlay network in point of the less redundancy between an overlay topology and an underlay topology [31]. In their paper, they referred to the path information and the hop count information in the granularity of AS and IP. In [32], Seetharaman et al. argued that a repository for routing services, high-level performance measurements and BGP information is needed for constructing an overlay-friendly underlying network. The repository is prepared for being used to provide optimized routes between peers by exploiting knowledge of underlying network characteristics.

2.3.2 Summary of the Requirements

As seen above, autonomic applications frequently utilize the network characteristics information of performance and structure, which also contains the identifier information. This can be explained by the fact that the performance and structure information has a great impact on the applications' behavior, whose purpose is to improve their quality and performance. And these kinds of information are obtained by network measurement, therefore, we can say that the network measurement matters to the applications a great deal. Additionally, because of selfish behaviors by overlay network applications, the overload and outage of network resources can occur [32, 33, 34, 35, 36], therefore, overlay network applications basically have to construct their own networks with care about this problem and many researchers have tackled a problem on the construction of overlay networks so as to improve their quality and performance and to suppress their selfish activities that disturb other applications.

After the confirmation of the importance of the network measurement for the autonomic applications, the next question is how the measurement should be performed. Traditionally, network measurement has been performed for the statistical analysis of network traffic and structure. And the analysis is basically performed after the mid-term or long-term collection of the network characteristics information. Like the public analysis data from CAIDA [37], the target data is generated after a certain period of collection, not immediately after the collection. However, the autonomic applications cannot wait for such a long term because they have to decide their future behavior with the collected network characteristics information. So one of the requirements for the application-oriented network measurement is the better responsiveness of collecting the target information. Another requirement is accuracy. If collected data lacks its accuracy, applications using this data will be misled on the decision of their behavior. For example, choosing a proximate node will fail if measured RTT data is wrong. Additionally, we will have to consider the possibility of the falsification of collected data. One more requirement is about the coverage of measurement. Since autonomic applications usually involve a number of nodes in multiple administrative domains, the measurement methodologies should have an ability for collecting the network characteristics information targeting these nodes.

The requirements for application-oriented network measurement is summarized as follows:

Responsiveness The collection of the network characteristics information should be done as rapidly as possible.

Accuracy The collected data should be accurate enough for not misleading applications. Usually the information must indicate the current status of a target when an application receives the information.

COVERAGE Methodologies for application-oriented network measurement should cover the range of nodes that an application cares. Basically the required coverage is all the nodes involved in an application service.

Considering these requirements, we present a fundamental design of a platform on which the application-oriented measurement is performed.

2.4 Designing an Application-Oriented Measurement Platform

As stated in Chapter 1, our approach for enhancing end-system capabilities is to provide an independent network service of the application-oriented measurement platform. We have several design choices for performing the application-oriented measurement. There are some reasons why we package the procedures of the application-oriented measurement into a service. Let us start with introducing them.

The first reason is that an independent network service does not disturb the end-to-end principle with which the current autonomic applications work. We now see that many large-scale autonomic applications are working with the end-to-end principle, and also see that their behavior can be improved with the application-oriented measurement. If the application-oriented measurement is provided as an independent network service, they just have to utilize it in the sequence of existing procedures. The term "independent" means that the service does not depends on any specific application or procedure, but completes its work within its territory, i.e., an independent service of network measurement just collects the requested network characteristics information and provides it to the application. The independent service does neither disturb the application's behavior nor force its utilization, therefore, such a service keeps the end-to-end principle.

The second reason is that an independent network service can serve any end systems without major changes in these systems. An independent network service is literally accessible from any network nodes on the Internet if they keep some rules for the access such as a communication protocol. This characteristic leads the ease of its utilization, and it is quite important because the application-oriented network measurement is usually performed in a wide area and involves a number of end systems. So the ease of the utilization has a big impact for motivating a number of the systems to utilize it.

The third reason is that an independent network service enables the easy and shared utilization of complicated measurement procedures. Even in the case of traditional and popular measurement methodologies such as ping and traceroute, writing a code for their procedures is just a burden for programmers because their true target is not the measurement but the provision of application services. Considering the measurement is just an assistant role, the measurement procedures must be provided in a convenient way. Additionally, some sophisticated measurement methodologies have their own complicated procedures, which keep ordinary programmers away. But once such the complicated procedures are implemented on an independent network service, everyone can commonly utilize them. In point of recycling the codes of measurement methodologies, an approach of an independent network service must be attractive.

Based on these ideas, we design a platform on which a service of the application-oriented measurement is constructed. We here call such a platform an application-oriented measurement

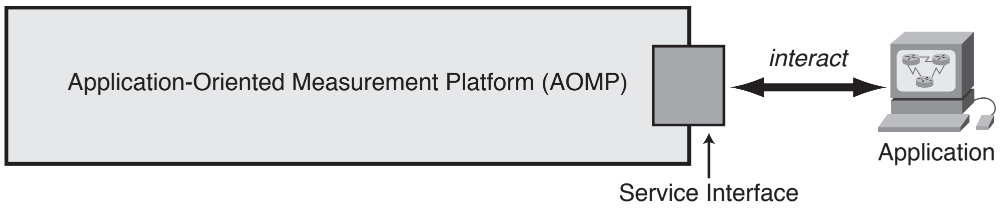


Figure 2.1: An application-oriented measurement platform as a service provider.

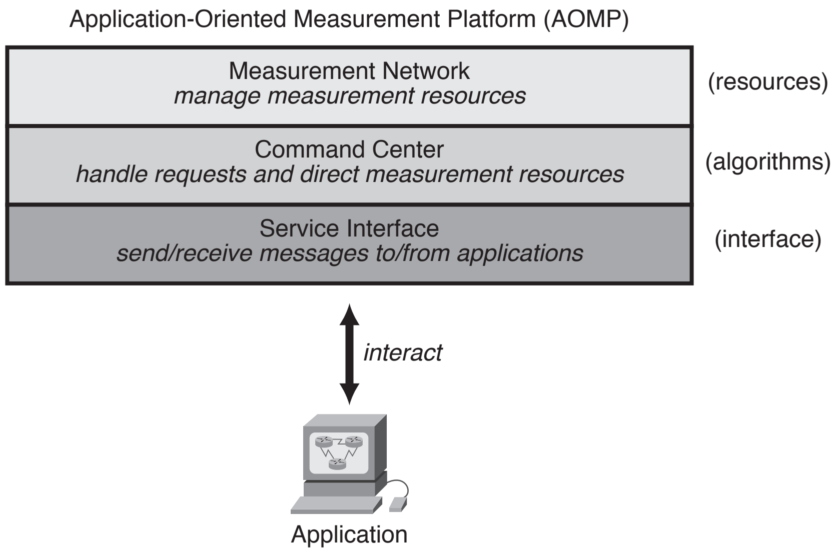

Application-Oriented Measurement Platform (AOMP)
Measurement Network
manage measurement resources
(resources)
Command Center
handle requests and direct measurement resources
(algorithms)
Service Interface
send/receive messages to/from applications
(interface)
interact
Application

Figure 2.2: Building blocks of an application-oriented measurement platform.

platform, or an AOMP in short. Through designing an architecture of the platform, we will have several design choices. With explaining the reasons of our choice, we gradually focus on the detail of the architecture.

First, we define two fundamental entities involved in the scenario of the utilization of AOMP. One entity is, of course, our AOMP, and the other is a network application that utilizes the AOMP. Since the AOMP provides a service of the application-oriented measurement, it must have an interface for the provision. We call the interface a service interface, and the application interacts with the AOMP through the service interface. Basically, the contents exchanged through the interface are request messages for network measurement from applications to an AOMP and collected network characteristics information from an AOMP to applications. This relationship is depicted in Figure 2.1.

Next, we divide the AOMP into two building blocks; one is a measurement network and the other is a command center. The measurement network provides several features of primitive measurement operations such as sending a specific packet and finding resources on the platform. The command center utilizes these features so that a target measurement algorithm correctly works. In other words, measurement methodologies are described in the command center and actual measurement procedures are performed on the measurement network. The service interface is attached to the command center, and the command center also interprets requests from applications and determine the behavior of the platform. Simply stated, the measurement network holds resources for measurement procedures, the command center has algorithms that leverage these resources, and the service interface is an gateway between applications and an AOMP. These are also explained in Figure 2.2. The reason why we divide the platform as above is that the management of resources and a neat command channel are quite important for an AOMP, considering the large-scale measurement like cooperative measurement methodologies. With these two layers, we can perform the intelligent behavior for measurement procedures, or in other words, we can manage the state information for network measurement. We can expect that this feature accelerates the implementation of sophisticated measurement methodologies that meet the requirements described in Section 2.3.

At last, we define the required functions for the respective layers. The functions are listed below.

1. Service Interface
   (a) The service interface receives a message from an application, and also sends a message to an application.
   (b) The service interface interprets a message from an application, and covert it to a proper format in which the command center can understand.
   (c) The service interface throws a message to the command center, and receives a message from the command center.
   (d) The service interface receives interprets a message from the command center, and convert it to a proper format in which an application can understand.
2. Command Center
   (a) The command center receives a message from the service interface, and also sends a message to the service interface.
   (b) The command center interprets a message from the service interface.
   (c) The command center decides how to utilize the measurement resources for achieving a goal requested in a message from the service interface.
   (d) Based on the decision, the command center controls the measurement resources through the functions provided by the measurement network.
   (e) The command center receives a message from the measurement network as a result of a utilized function. The message usually contains the measurement data.
   (f) For the application's goal, the command center calculates the target values from the measured data contained in the messages from the measurement network.
   (g) The target data is converted to a proper format in which the service interface can understand.
   (h) A programmer can write his or her measurement and computation algorithms in the command center.
3. Measurement Network
   (a) The measurement network provides primitive functions of network measurement for the command center.

(b) The measurement network accepts a command from the command center, and perform a procedure as preliminarily defined.

(c) The measurement network provides the result of a performed procedure as a message to the command center. The format of the message can be understood by the command center.

A notable point is 3a in the above list. At this moment, we define several primitive functions of network measurement considering the actual measurement procedures. The primitive functions that should be implemented on the measurement network are summarized as below.

- Generate and send a packet
- Receive and read a packet
- Find other measurement resources
- Exchange messages among monitoring nodes
- Share the data among the measurement resources

In order to perform a measurement procedure, the handling of packets is necessary. The first two items are defined for this reason. The third, fourth, and fifth items are defined especially for the cooperative measurement. In cooperative measurement, monitoring nodes basically cooperates by communicating with other nodes and sharing the collected data. These items are required for such procedures.

2.5 Summary

In this chapter, we defined the fundamental requirements for the application-oriented network measurement, considering the current form of the utilization of the network characteristics information and actual measurement methodologies. Since the requirements for the application-oriented network measurement was not yet defined explicitly, we expect that our analysis issues a guideline for such measurement. Then we presented a design of the architecture of an application-oriented measurement platform (AOMP). Our design is unique in point of the consideration for implementing cooperative measurement methodologies on a platform. For the verification of the reasonability of our design, we construct an actual AOMP and evaluate its capability in the following Chapter.

Chapter 3

N-TAP: An Application-Oriented Measurement Platform Built on a Role-Based Peer-to-Peer Network

In the previous chapter, we described a fundamental concept of application-oriented network measurement and summarized some its requirements. Based on them, we now focus on an actual system for application-oriented network measurement. For constructing such a system, we will have a lot of design choices. Revealing our choices, we introduce a distributed measurement platform called "N-TAP," as an instance of AOMP.

A key feature of N-TAP is its emphasis on constructing the system in a distributed manner. Considering the variety of measurement nodes for the Internet observation, such as their stability and performance, N-TAP's measurement nodes are connected as sparsely as possible. This means that the nodes can arbitrarily perform their tasks, but that we have to prepare a coordination scheme for their activities at the same time. As one solution to this concept, we adopt a role-based peer-to-peer network for N-TAP's measurement network, and implemented some fundamental features for an AOMP service on it, according to the AOMP design presented in the previous chapter.

The fundamental features include the shared storage and agents search. While N-TAP acts as an independent measurement service for autonomic applications, it also works as a platform on which cooperative measurement methodologies can be implemented. We also highlight the formation of measurement networks and provide a first look at measurement activities performed on those networks. From the viewpoint that the rapid provision of network characteristics information is an important requirement for an application-oriented measurement platform, we conduct a performance evaluation of the essential features of N-TAP through the experiment on PlanetLab. Based on this evaluation, we also discuss the tactics for improving the measurement activities on a peer-to-peer measurement platform.

3.1 Design Choices

We present the design of N-TAP and the reasons for the design choices in the following subsections. We first outline the concepts of N-TAP. Based on the concepts, we design the architecture of N-TAP and describe how its components work. The last subsection presents the details of our implementation of N-TAP.

3.1.1 Concepts

The design concepts for N-TAP are broadly divided into three components: independent service, platform for cooperative measurement, and tactical measurement.

Independent Service

The first concept is "independent service", whose purpose is to construct N-TAP as an independent service in order to increase the ease of handling network characteristics information for a variety of applications. Currently we don't have a simple way for collecting network characteristics information, which will make application developers avoid to utilize such information. For a solution to this problem, we choose to abstract common measurement procedures to one framework. If the framework can provide a simple interface that can invoke complicated measurement procedures, we can expect that the developers' burden will be reduced by using it. Additionally, with an independent measurement service, we can expect that some of network characteristics information collected for one application will be reusable in other applications, which will result in the reduction of measurement overhead. This concept is naturally derived from the one of AOMP.

Based on this concept, we now define some requirements. First, N-TAP must provide an interface for interacting with applications, and the interface should weakly connect N-TAP and the applications so that a variety of applications can utilize N-TAP. At the same time, the communication protocol and the formats of network characteristics information must be defined for general use. This is for expanding the range of people who can utilize N-TAP. Secondly, N-TAP must abstract the details of its measurement procedures, i.e., N-TAP must be simple to use from outside the system. This allows applications to obtain network characteristics information very easily, not unlike getting content from a web server.

Platform for Cooperative Measurement

The second concept is "platform for cooperative measurement," which accelerates the deployment of sophisticated methodologies by providing fundamental features for such measurement. Compared to classical measurement methodologies, cooperative measurement methodologies are more attractive from the viewpoints of lower measurement overhead, the wide variation of collected network characteristics information and so on. As described later in Section 3.5.1, communication among measurement nodes and the sharing of collected data are often required as the common features in cooperative measurement. The provision of a measurement platform on which we can utilize these features will make the implementation of such methodologies easier. If a number of cooperative measurement methodologies are implemented on N-TAP, they increase its measurement capability, especially for large-scale measurement like on overlay networks. Therefore it will appeal to developers to utilize network characteristics information in their applications. This will also appeal to the researchers of measurement methodology because N-TAP can be a large-scale testbed of cooperative measurement in the actual network environment.

As the common features stated above, N-TAP must prepare the mechanisms for communication channels among measurement nodes and shared databases of collected network characteristics information. The communication channel allows a measurement node to exchange messages with other measurement nodes. For example, this feature will be used for the synchronization of measurement procedures, or a request to other nodes for obtaining specific data. The shared database realizes the utilization of network characteristics information that other nodes collected. These common features can be expected to help the implementation of cooperative measurement methodologies on N-TAP and make them available for applications.

Tactical Measurement

The last concept is "tactical measurement," which is important as an application-oriented measurement platform. Since the requirements for collecting network characteristics information depend on each application, N-TAP must interpret such requirements carefully and make a decision on the action of collecting requested data. For example, we should not provide only the topology data which was collected 10 years ago to an application that intends the reconstruction of its overlay multicast tree because the topology may not reflect the current topology. However, if such topology data have been collected over a long period and we can estimate that the topology might not change from the long-term observation, the old data still counts. Especially for autonomic applications, the provision of network characteristics information should be done rapidly as stated in Section 2.3.2.

As just described, N-TAP must prepare a mechanism that enables applications to specify their requirements for network characteristics information such as the freshness of collected data. Based on such requirements, N-TAP has to make the tactics for the collection of desired characteristics information including the choice of measurement methodologies with the consideration of their properties like accuracy, overhead and responsiveness.

3.1.2 Architecture

Based on the concepts described in Section 3.1.1, we illustrate the design of N-TAP's architecture in this section. First we present the outside view of N-TAP and the interaction among N-TAP and other entities. Then we show the inside view of N-TAP and the connection among the components of N-TAP.

Outside View

We first define four entities that appear in the scenario of the utilization of N-TAP. The first entity is the "N-TAP agent", a program that runs on certain nodes on the Internet. We call a node on which the N-TAP agent runs the "N-TAP node" as the second entity. The N-TAP agent performs all kinds of procedures related to N-TAP; in other words, N-TAP is constructed with only the N-TAP agents. We assume, in principle, that any node of the overlay application that utilizes N-TAP runs an N-TAP agent in its local environment and requests network characteristics information to its local agent. The third one is the "application", which requests network characteristics information to N-TAP for its own purposes. The last one is the "general network element", which is a component of the Internet such as nodes and links.

Figure 3.1 describes the relationships among these entities. In this figure, each object depicts one of the entities and an arrow means the direction of actions; a "request" arrow from an object A to an object B means that A makes requests to B. An N-TAP agent accepts the application's request for obtaining network characteristics information from both the local and remote sides and

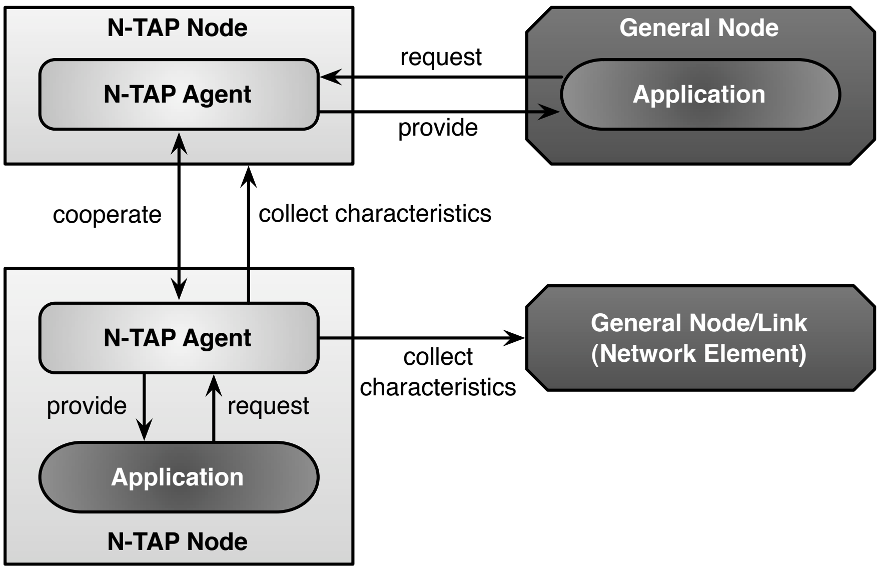


Figure 3.1: Relationships among the entities that appear in N-TAP.

provides network characteristics information to the application as its response. Based on the request, N-TAP performs measurement procedures to collect network characteristics information. Its measurement targets are general network elements including N-TAP nodes. Moreover, an N-TAP agent cooperates in measurement with other N-TAP agents if necessary.

Inside View

The N-TAP agent has four components inside itself as shown in Figure 3.2: network characteristics database, network characteristics provider, network characteristics collector and N-TAP network manager. The network characteristics database is a local repository of collected network characteristics information. Each agent has its local database and stores collected network characteristics information into it. The network characteristics provider is an interface between an N-TAP agent and other applications. The interface provides APIs by which applications can obtain network characteristics information from N-TAP. Applications can request the details and conditions of required network characteristics information (e. g., the kind of network characteristics information) through the interface. After accepting the request from the application, this component decides how to collect the requested network characteristics information and then searches the requested network characteristics information in the network characteristics database or requests the collection to the network characteristics collector based on its decision. We describe the details of such decision making later. The network characteristics collector performs measurement procedures in order to collect the network characteristics information requested by the network characteristics provider. One collector can communicate with other collectors for cooperative measurement. The N-TAP network manager is responsible for forming the N-TAP network, a measurement overlay network among N-TAP agents. On the N-TAP network, any agent can access the network characteristics information collected by other agents; in other words, N-TAP agents can share their collected network characteristics information. It also searches the N-TAP agents that other components (collector or provider) need for cooperation

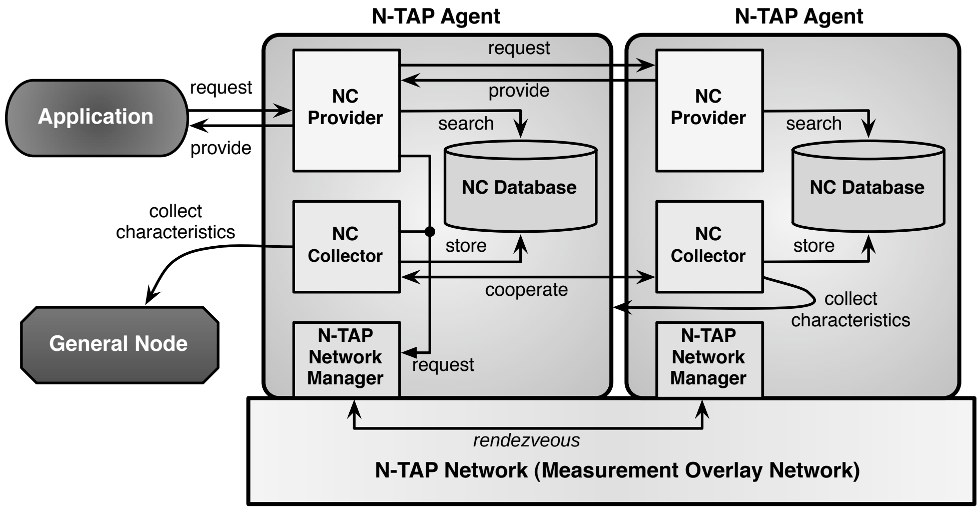


Figure 3.2: Architecture of an N-TAP agent (NC is an abbreviation for network characteristics).

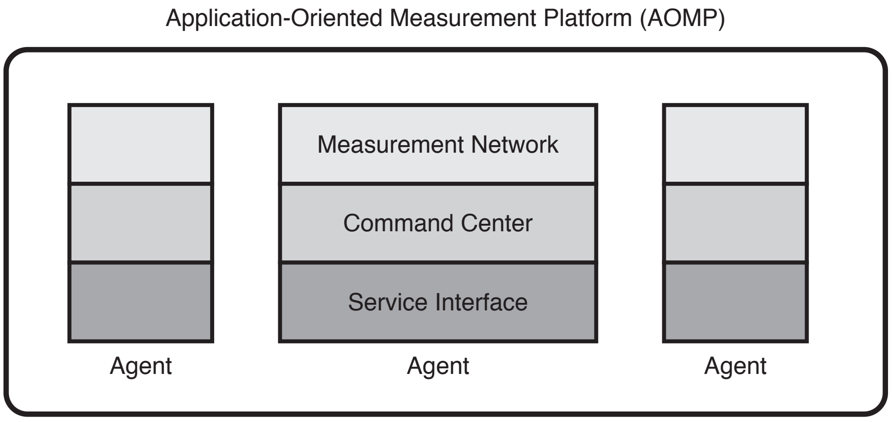


Figure 3.3: Correspondence between an N-TAP agent and the AOMP components.

and sets up a place for their rendezvous. With these components, N-TAP realizes the independence of its service and cooperative measurement remarked in Section 3.1.1.

3.1.3 Correspondence to the AOMP Design

Here we review the AOMP design introduced in Section 2.4 and check the correspondence between this and the N-TAP's design. According to the design of N-TAP, the N-TAP agents play all the roles of the service interface, the command center, and the measurement network. That is, we vertically divide the AOMP depicted in Figure 2.2, and assign each block to perform as an N-TAP agent, as shown in Figure 3.3. One of the advantages of adopting such the vertical division of the functions is that we just have to provide a unitary program of the N-TAP agent for the deployment. This will decrease the complexity of the entire system, and then we can expect more wide-spread deployment. Another advantage is that this approach suits a style of the large-scale measurement such as cooperative measurement. As peer-to-peer network applications, cooperative measurement methodologies also expect an autonomic behavior of its monitoring node,

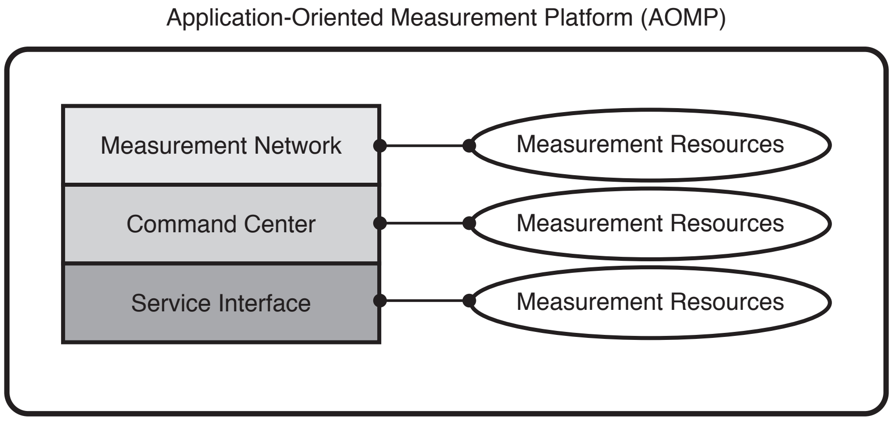


Figure 3.4: Horizontal division approach of the AOMP functions.

and basically the responsibility for each node is equal among the nodes. Therefore, we suppose that this approach is reasonable for the large-scale Internet observation.

We could also adopt a contrastive approach, i.e., the horizontal division depicted in Figure 3.4. In that approach, different measurement resources in the platform are assigned for the respective functions. This will allow the easy trace of the procedures in the platform and provide a clear border of responsibilities. However, we have to have a burden for assigning measurement resources to specific roles, and we can expect that it is quite difficult in case of the large-scale platform with massive resources. Additionally, by dividing the system horizontally, its users or programmers for the command center may face a problem of grasping such assignments. Equipping one program with all functions has an advantage of the simplicity coming from uniformity. For these matters, our design prefers the vertical division approach.

3.1.4 Measurement Network Models

In this section, we first define the components of a measurement network and how they work and interact with other entities. In Section 3.1.4, we describe two existing models of measurement networks — centralized and pure peer-to-peer models. We also refer to a hybrid measurement network model in the same section. Finally, we propose a methodology to allow shifting a measurement network between these models, and we describe its implementation on an actual measurement system.

Components of the Measurement Network

A measurement network is a network in which measurement procedures are performed according to predefined sequences. Here we define the entities that appear in a measurement network and its operation.

The first entity is a “monitoring node,” which performs measurement procedures in order to collect network characteristics information. The second entity is a “management node,” which is responsible for coordinating other entities so that the intended measurement can be performed. For example, the management node inspects and updates “management information,” such as the list of monitoring nodes, and commands some of the monitoring nodes to perform measurement procedures. Collectively, we call a system that is composed of management information

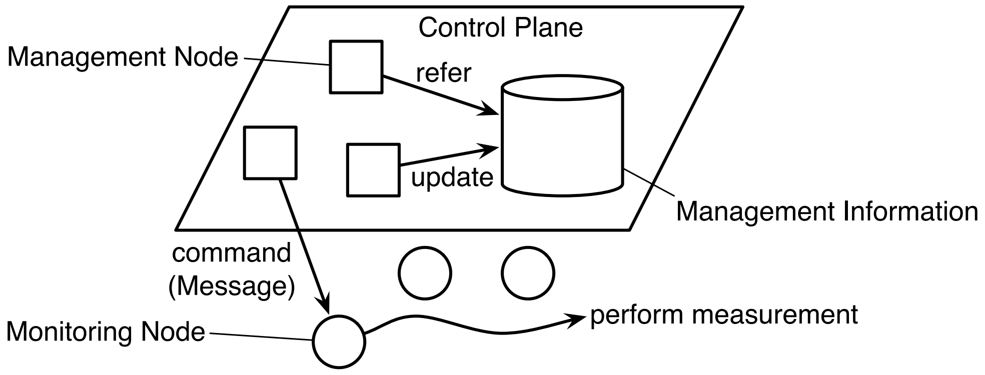


Figure 3.5: Components in the measurement network and relationships among them.

and management nodes a "control plane." A control plane is, so to speak, an entity where decisions for measurement procedures are made. "Control messages" are exchanged among the monitoring and the management nodes to achieve the intended measurement features. The control messages include a measurement command to the monitoring nodes and the node list in the measurement network, but do not contain the network traffic derived from the measurement procedures themselves. We note that one physical node may simultaneously play the roles of both management and monitoring. Figure 3.5 shows the relationship among the entities described in this paragraph.

Three Types of Models

Existing measurement networks are categorized mainly into two models — centralized and pure peer-to-peer models. In the centralized model, one management node or a cluster of replica nodes manages all of the management information and issues control messages to the monitoring nodes. On the other hand, in the pure peer-to-peer model, all of the nodes take the roles of both monitoring and measurement. Therefore each node has to maintain the measurement network and has also to perform the necessary measurement procedures. The merit of the centralized model is that the responsibilities of the respective nodes are clear, and it is easy to follow the sequence of measurement operations. At the same time, a central management node has to tolerate a heavy load caused by all the management operations, otherwise the measurement system will not function. In the pure peer-to-peer model, we can distribute such loads to all nodes; hence this model is considered appropriate for a large-scale measurement system. However, a frequent change in the state of the measurement network, such as nodes joining and leaving, will result in poor stability of the control plane. These trade-offs are also discussed as a general problem existing between centralized and peer-to-peer systems.

As a middle course between these models, we now consider a hybrid measurement network model. In the hybrid model, management operations are divided among some management nodes, while other nodes behave as monitoring nodes. The difference between the hybrid model and the centralized model is that, in the hybrid model, multiple management nodes each perform a different management operation, whereas the management operations are not clearly divided in the centralized model even if there are multiple management nodes. By adopting this model, we can expect to moderate both the load concentration and the instability of the measurement

```
$a$.storeCollectedData($collectedData$) {
    FOR each $index$ of $collectedData$ DO
        {Generate a data entry.}
        $id \leftarrow a$.generateEntryId($index$)
        $entry \leftarrow a$.formatDataEntry($id$, $collectedData$)
        {Store the entry in the local database.}
        $a$.storeDataEntryInLocalDatabase($entry$)
        {Store the entry in the databases of other agents.}
        $a' \leftarrow a$.findResponsibleAgent($id$)
        IF $a'$ is not $a$ THEN
            $a'$.storeDataEntryInLocalDatabase($entry$)
        END IF
    END FOR
}
```

Figure 3.6: Algorithm for data storage.

network, which are the problems in the first two models. This model is similar to that of the Kazaa [38] network, in which stable nodes (called "super nodes") construct an overlay network in a peer-to-peer manner, and ordinary nodes join the overlay network through the super nodes.

3.1.5 Implementation and Experience

In this section, we present the mechanism of N-TAP's fundamental features including the shared database and the communication channels. We also introduce a tactic called the "local-first and remote-last" rule, which N-TAP adopts to improve the responsiveness on responding to the requests from applications and reduce measurement overhead.

Shared Database

As stated in Section 3.1.1, the feature of shared database is essential for cooperative measurement. N-TAP forms a shared database where N-TAP agents can store and retrieve the collected data with some indices.

In order to form the measurement overlay network called "N-TAP network", we utilize the technique of Chord [19], which is one implementation of the distributed hash table (DHT). Since the size of collected network characteristics information is massive in the case of large-scale measurement, we choose Chord to construct the shared database for distributing storage costs to each node. In the N-TAP network, N-TAP agents and database entries have their own identifiers whose length is 128 bits. An agent ID locates the agent in the Chord ring and an entry ID decides which nodes store the entry in their own local databases. To generate a random ID from a data stream, we apply the SHA-1 cryptographic hash function to the stream and use the first 128 bits of its 160-bit result. The N-TAP agent uses SQLite [39] for constructing a local database.

N-TAP constructs a shared database by using the nature of Chord, i. e., the N-TAP network is essentially a distributed database of network characteristics information. Figure 3.6 presents an algorithm for storing collected network characteristics information. After an agent collects network characteristics information, the collected data (*collectedData* in the figure) is formatted as a database entry (*entry*) with an entry ID (*id*) obtained from the generateEntryID() function, which generates a random ID as shown in the previous paragraph. The indices of each kind of network characteristics information are preliminarily defined in order to let the collected data accessible. Then the entry is stored in the local database of the agent that collected (agent a) and in the local databases of the nodes (a') that are responsible for the entry ID obtained from the respective indices in the collected data. The findResponsibleAgent() function actually utilizes a essential feature of Chord that finds a successor of a specified ID in the Chord ring. To retrieve a data entry from the shared database, an N-TAP agent just has to search a node that is responsible for the entry ID and request the node to send back the data entry.

For example, if a node A collects the RTT between nodes A and B and the indices of RTT data are source IP address and destination IP address, the data is stored in a node A itself and the nodes that are responsible for generateEntryId(IP(A)) and generateEntryId(IP(B)) in the Chord ring, where $IP(N)$ denotes the IP address of a node N. This data can be retrieved by specifying its entry IDs that can be generated from IP(A) or IP(B).

Communication Channels

The feature of communication channels is also realized on the N-TAP network by handling the information of N-TAP agents as entries in the shared database. Each agent puts the information about itself into the shared database after it joins to the N-TAP network in order to find each other for cooperation. They use their IP addresses, netmasks, fully qualified domain names (FQDNs) and so on, as the indices of the put information, and N-TAP network manager can search other agents with these indices.

"Local-First and Remote-Last" Rule

In this section, we describe the process of decision making on measurement activities conducted by the network characteristics provider. In order to reduce measurement cost and improve responsiveness on collecting network characteristics information, the process adopts the basis of "local-first and remote-last." The sequence of its procedures is described in Figure 3.7. This rule categorizes the situation of requested network characteristics information into the following four types if the data can be collected by N-TAP:

1. An agent has requested data: the data can be obtained from its local database.
2. An agent can measure the data by itself.
3. Other agents already have the data: the data can be obtained from the shared database.
4. The data can be measured by (cooperating with) other agents: the local agent issues a request to other agents.

N-TAP prioritizes the actions for respective situations as listed above to minimize the turn around time for applications' requests. First, the network characteristics provider searches the network characteristics information that meet the request from application in the local database. If the requested data cannot be obtained from the local database, it checks whether or not the local agent can measure the data by itself. If the agent can, it requests to the network characteristics collector

```
$a$.getNetworkCharacteristics($requestedData$) {
   {Search the requested data in the local database.}
   $result \leftarrow a$.searchInLocalDatabase($requestedData$)
   return $result$ IF $result$ is not NIL
   {Measure the data by itself if possible.}
   IF $a$ can measure $requestedData$ THEN
      $result \leftarrow a$.measureNetworkCharacteristics($requestedData$)
      return $result$ IF $result$ is not NIL
   END IF
   {Search the data in the shared database.}
   $result \leftarrow a$.searchInSharedDatabase($requestedData$)
   return $result$ IF $result$ is not NIL
   {Forward the measurement request to other agents.}
   $candidates \leftarrow a$.searchCandidates($requestedData$)
   FOR each $a'$ in $candidates$ DO
      $result \leftarrow a'$.getNetworkCharacteristics($requestedData$)
      return $result$ IF $result$ is not NIL
   END FOR
   {Cannot obtain the requested data.}
   return NIL
}
```

Figure 3.7: Algorithm for collecting network characteristics information.

to collect the data; otherwise, it searches the data in the shared database. If the data doesn't exist in the shared database, by asking the N-TAP network manager, it tries to find other agents that have or can collect the data. If one or more agents are found, it requests the (cooperative) measurement to the network characteristics provider of theirs. Or if not, it tells the unavailability of the requested data to the application. The provider that accepted the request from another agent performs the same decision-making process. However, it no longer forwards the request to other agents and immediately replies in the case of unavailability. Through the simple steps described above, the network characteristics provider decides what to do for collecting the requested network characteristics information.

Agent's Roles

Our key idea is the division of the agent's roles into core agent and stub agent. The core agent, which corresponds to the management node, constructs the measurement overlay network, called the N-TAP network, as conventional agents did: it maintains its own routing table in the Chord ring and stores some of the shared data in a local database as a part of the shared database. The core agent also performs measurement as a monitoring node if necessary. The stub agent, which is equivalent to the monitoring node, does not perform the operations related to the construction of the N-TAP network. For joining the N-TAP network, the stub agent inserts its information in the shared agent list so that other agents can find it. It performs measurement only when a core agent sends a request to it or when it knows that the measurement procedures that are requested directly from applications should be done by itself. In the case that the stub agent needs to do the operations related to the N-TAP network, it sends a request to one of core agents, and the core agent responds to the request. For example, suppose that a stub agent wants to find a core agent that is responsible for a specified ID in the Chord ring so as to retrieve a shared data entry that has this ID; the stub agent asks a core agent to find the responsible agent, and the core agent performs the procedure of finding it. After the core agent obtains a result, it sends the result to the stub agent. In this way, even a stub agent, which does not perform the management procedures for the N-TAP network, can know the state of the N-TAP network.

By adopting the scheme of core and stub agents, we can also easily form the centralized and pure peer-to-peer measurement networks. Figure 3.8 shows the transitions of measurement networks according to the allocation of the respective numbers of core and stub agents. Now we have N agents, and C of N agents are assigned as core agents; i. e., $S\ (= N - C)$ agents are stub agents. The N-TAP network where $C = 1$ is equivalent to a centralized measurement network because all of the management information is concentrated in one core agent. If we take the value of $C = N$, all of the agents are core agents; therefore the N-TAP network in this situation is a pure peer-to-peer network, which is same as the conventional N-TAP network. In case of $C = i\ (2 \leq i \leq N - 1)$, we can regard the N-TAP network as the hybrid measurement network.

As described in this section, we can now have three types of measurement networks on the actual measurement system. In the following sections, we investigate the basic characteristics of these measurement networks.

Other Implementation Topics

As collection methods, N-TAP currently can *ping* to measure the RTT between an N-TAP node and another node, and can also *traceroute* to obtain the IP topology from an N-TAP node to an-

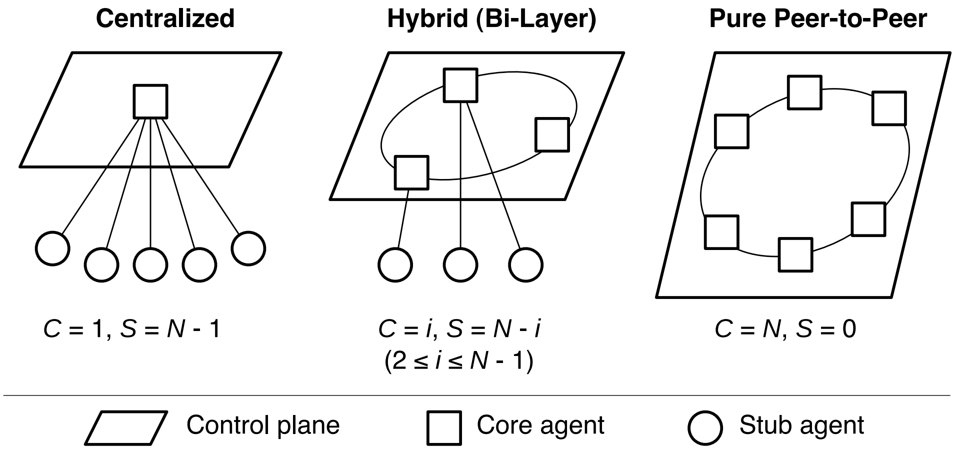


Figure 3.8: Measurement network formations with the scheme of core and stub agents ($N = 6, i = 3$).

other node. Furthermore, since an N-TAP node can refer to the collected data by other N-TAP nodes or request *ping* and *traceroute* to other nodes through N-TAP network, the N-TAP node can also obtain the RTT and IP topology among other nodes, and the IP topology whose start point is not the N-TAP node itself. This is a primitive method of cooperative measurement from the standpoint that one node can obtain the network characteristics information that cannot be collected by the node itself.

For the protocol of the interaction between an application and the N-TAP agent, we adopt XML-RPC [40] because of its widespread deployment and description capability. In order to request the network characteristics information, the application calls the agent's methods for collecting the target data while specifying the kind of network characteristics information and certain conditions for the data. The collected data is stored with additional information such as UNIX time stamp, the ID of the agent that collected the data, its collection method, and so on. They are the criteria for judging whether the data can show the actual state of network entities, and the judgment depends on the conditions offered by an application.

At this time, we have a prototype implementation of N-TAP, which is based on the design described in this section. It works on some platforms including FreeBSD, Mac OS X and Linux on PlanetLab [41].

Experience

Here we describe one simple scenario of retrieving network characteristics information from N-TAP. There are three N-TAP nodes running one N-TAP agent on the respective nodes: nodes A (IP address: 192.168.1.10), B (192.168.2.20) and C (192.168.3.30). The application running on the node C wants to know the RTT between the nodes A and B, which is collected within 300 seconds, so it requests to the local N-TAP agent on the node C by calling the method for obtaining the RTT data (ntapd.getNetworkCharacteristics.round-TripTime.IPv4). The request message is shown in Figure 3.9. Based on the process of decision making described in Section 3.1.2, the agent on the node C first searches the data that meets the request (RTT from A to B collected

• Request Message

```
<?xml version=”1.0” encoding=”UTF-8”?>
<methodCall>
<methodName>ntapd.getNetworkCharacteristics.roundTripTime.IPv4</methodName>
<params>
<param><value><struct>
<member><name>sourceIPv4Address</name><value><string>192.168.1.10</string></value>
</member>
<member><name>destinationIPv4Address</name><value><string>192.168.2.20</string>
</value></member>
<member><name>freshness</name><value><i4>300</i4></value></member>
</struct></value></param>
</params>
</methodCall>
```

• Response Message

```
<?xml version=”1.0” encoding=”UTF-8”?>
<methodResponse>
<params>
<param><value><struct>
<member><name>timestamp</name><value><i4>1122334455</i4></value></member>
<member><name>collectionMethod</name><value><string>ICMPEcho</string></value>
</member>
<member><name>roundTripTime</name><value><i4>12569</i4></value></member>
</struct></value></param>
</params>
</methodResponse>
```

Figure 3.9: Example of request and response messages for obtaining RTT data.

within 300 seconds) in its local database. In this case, we assume the agent on the node C cannot find such data in the local database. Since the node C cannot measure the RTT between nodes A and B by *ping*, the agent on the node C decides to search the data in shared database. Unfortunately, such data is not in the shared database, so the agent on the node C confirms the existence of the N-TAP node A by the list of N-TAP agents in the shared database, and forwards the request to the agent on the node A in order to measure it. The agent on the node A performs RTT measurement between the nodes A and B, then replies to the agent on the node C with the result of the measurement. Finally, as shown in Figure 3.9, the agent on the node C gives the requested data to the application with its response message.

3.1.6 Reasons for Using a Role-Based Peer-to-Peer Network as a Measurement Network

There are some reasons for our choosing a role-based peer-to-peer network as the N-TAP's measurement network. Here we describes the reasons.

The first reason is that a peer-to-peer based node management can naturally involve end systems. As stated, autonomic applications basically require the network characteristics information related to the end systems involved in application's service. To obtain such information, we usually need to let these end systems collect the required network characteristics information. The peer-to-peer network composed of the end systems can meet this coverage requirement.

The second is that we can expect to ensure the scalability with a peer-to-peer based system. Since the autonomic applications are often deployed on a large scale, more number of measurement targets and measurement nodes are also involved in the measurement service. We need to let our service work correctly even on a large scale with the consideration for accuracy and responsiveness. Since a peer-to-peer based system has less failure points and management points, we expect that it has a merit on scalability.

The third is its small deployment cost. With a peer-to-peer based system, we do not necessarily need to prepare a fixed resource for an infrastructure of a system. This reduces the management cost for the overall system and is effective especially in case of a large-scale system. If more end systems are involved in our system, we can extend the coverage of measurement, therefore, the we take notice of the small deployment cost of a peer-to-peer based system.

Additionally, the sparse connection between N-TAP agents on the role-based peer-to-peer network, namely that the agents interact with each other through the shared database, provides the arbitrary property to the agent's behavior. N-TAP is designed to decrease the direct interaction among the agents by replacing such procedures with the shared storage feature like the communication channels. Owing to this design, the agents less need to care the status of other agents, for example, whether an agent is alive or not. The merit of the sparse connection will also appear in Chapter 4.

3.2 Performance Evaluation of the Core Network

So far we have pointed out the common features for cooperative measurement and presented the mechanisms that realize these features on a measurement overlay network. As stated, the rapid provision of network characteristics information is required for an application-oriented measurement platform. In this section, we investigate the performance of N-TAP's fundamental features to prepare for discussing tactics for measurement activities. Our experiment was performed with the PlanetLab nodes that were randomly chosen. One N-TAP agent ran on each node, and a ran-

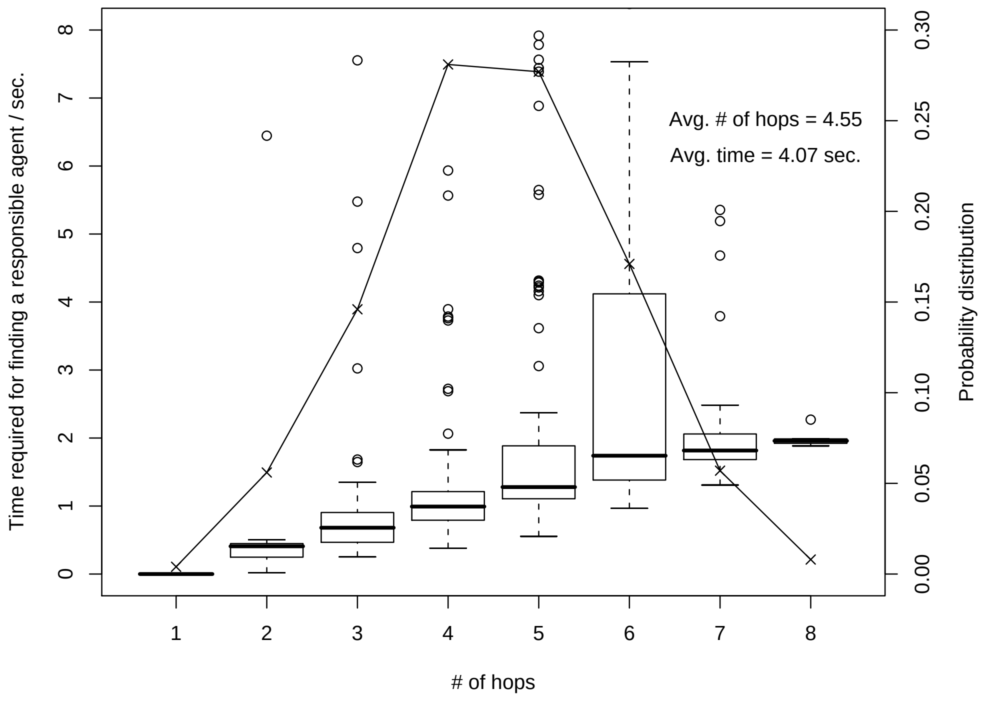


Figure 3.10: Turn around time for the requests of finding a responsible agent.

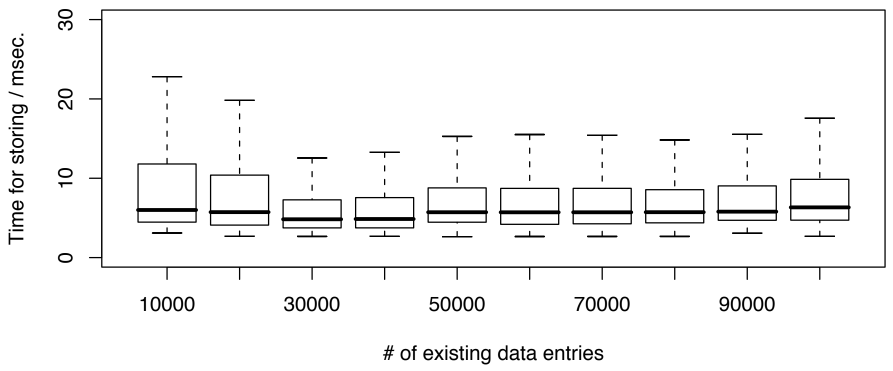
dom agent ID without overlaps was assigned to each agent.

3.2.1 Measurement Overlay Network

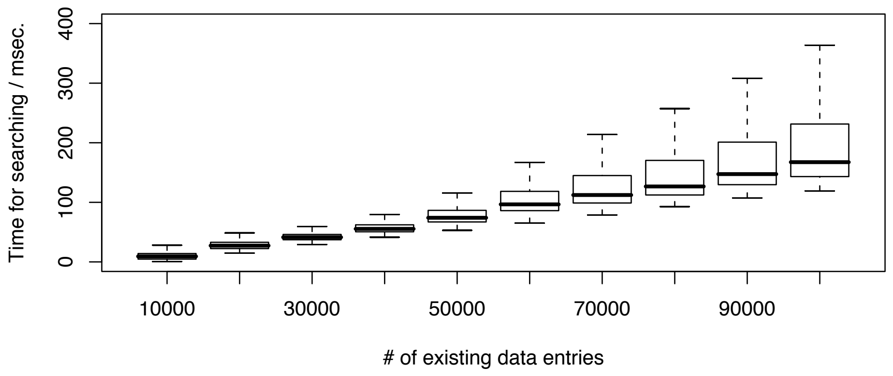
N-TAP forms its own measurement overlay network with the technique of Chord [19], which is one implementation of the distributed hash table (DHT). An essential feature of the DHT systems is to find out one or more responsible nodes for a specific ID in their ID spaces. In the case of Chord, finding a successor of a specific ID corresponds with such feature. So first in this section, we investigate the performance of this function, which affects functions in the upper layer.

We randomly chose 128 PlanetLab nodes and let N-TAP agents run on these nodes, namely a Chord-based network with these 128 nodes was formed. On this measurement overlay network, we executed a procedure of finding a responsible agent for a random ID on a randomly chosen node and measured required time for the procedure. After 1000 times repetition, we obtained a performance trend shown in Figure 3.10. In this figure, "# of hops" means the number of agents that received a request of finding a responsible agent recursively, i. e., if the number is 1, an original agent didn't need to ask other agents to find a responsible agent, and if the number is 4, the 4th agent in forwarding a request replied the information of a responsible agent. The boxplot shows the distribution of required time to receive an answer to a request, and the line graph shows the probably distribution of the number of hops.

The average number of hops is around $1 + \frac{1}{2}\left(\log_2 N\right)$ where N is the number of overlay nodes (agents), which coincides with the simulation result shown in [19]. The median of turn around time exhibits linear trend as the number of hops increases. This can be understood by considering that forwarding a request averagely takes same time at each agent. Additionally, we



(a) Store



(b) Search

Figure 3.11: Turn around time for handling a local database.

suppose some points plotted far from whiskers are caused by temporal high load on the PlanetLab nodes (as a shared experimental infrastructure) derived from other users' experiments. From these results, we can find that the required time for finding a responsible agent increases linearly with the increase of the logarithm of the number of N-TAP agents.

3.2.2 Local Database

Each N-TAP agent has a local database that is a part of the shared database on the measurement overlay network. Next we measured the time-based performance of storing or searching in a local database, which is another factor that influences the functions in the upper layer.

As the performance of storing a data entry in a local database, we measured the required time for storing one entry of RTT data between two nodes. As for searching, we measured the required time for searching an entry that has a randomly chosen ID. Figure 3.11 shows the distribution of required time for each procedure when the local database already contains from 10000 ($n - 1$)

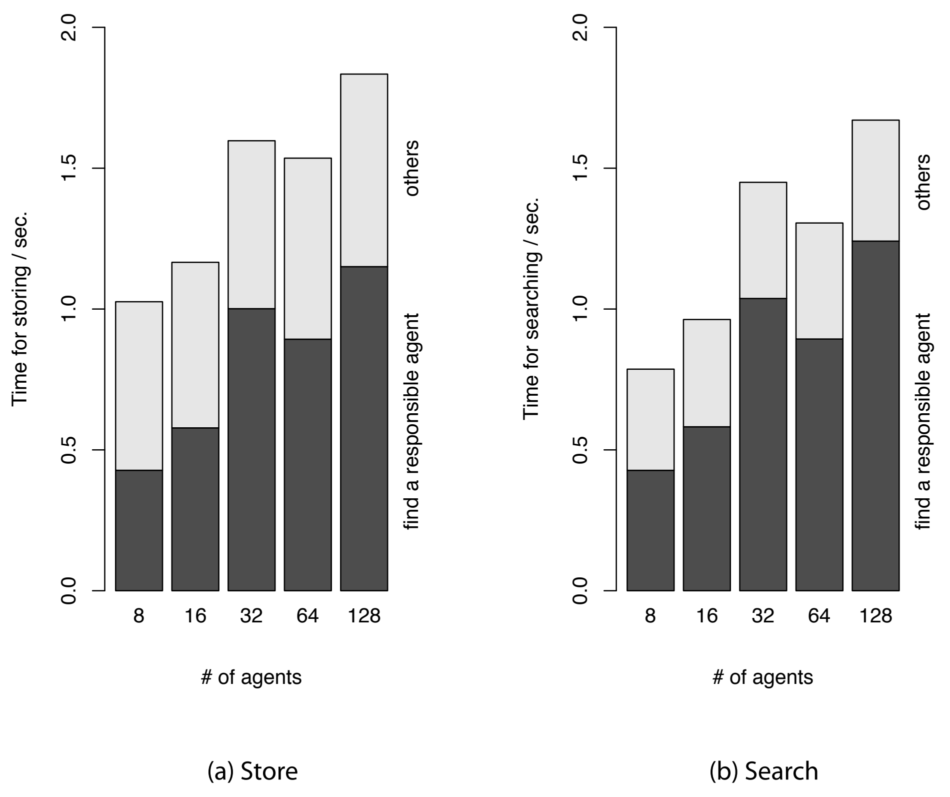


Figure 3.12: Turn around time for handling the shared database.

to 10000n data entries ($n = 1, 2, ..., 10$) as represented in the X-axis. As we can see in the figure, turn around time for storing a data entry keeps approximately constant regardless the size of a local database within this range. On the other hand, turn around time for searching data entries increases linearly with the increase of the database size. The median values of turn around time for storing and searching where $n = 10$ are respectively 6.33 ms and 167 ms, and we can point out that the cost for searching a data entry in a large local database is much higher than one of storing a data entry from the aspect of required time.

3.2.3 Shared Database and Communication Channels

The procedure of storing or searching data entries in the shared database is mainly composed of two fundamental functions. The N-TAP agent first has to find a responsible agent for the data entry ID and then communicate with the found agent to put or retrieve the data entries.

We measured the turn around time for storing or searching data entries with varying the number of agents (Figure 3.12). In the respective cases where $N = 8, 16, 32, 64, 128$, we chose an agent sequentially and invoked the procedure of storing or searching one data entry of RTT information in the shared database on the agent. The graphs show the median values of turn around time for storing or searching procedures after 1000 times repetition of this procedure. The dark gray part of each bar is the required time for finding a responsible agent for a data entry and the rest is the required time for other procedures including establishing a connection with

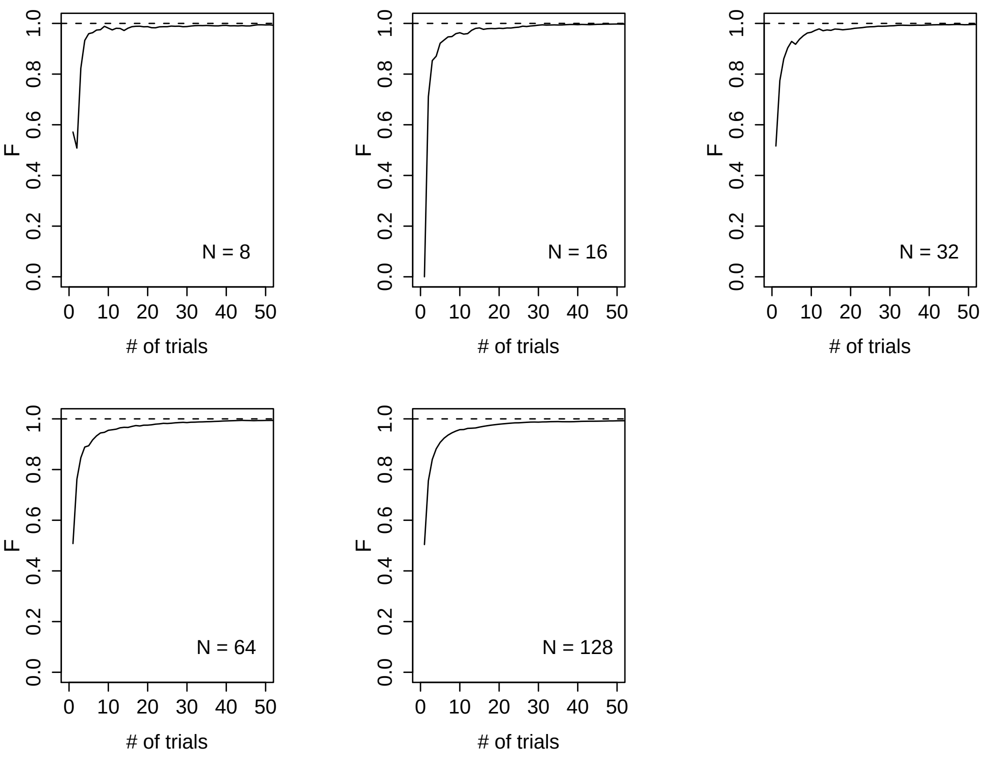


Figure 3.13: Fairness of the distribution of data entries.

the agent and sending or receiving data entries. From this result, we can find that the required time for finding a responsible agent increases as stated in Section 3.2.1 but the time for other procedures just shows slight change against the increase of the number of agents. Consequently the required time for finding a responsible agent becomes dominant in the overall required time as the number of agents grows. If the size of the measurement overlay network is constant and the number of data entries continues to increase, the required time for the procedures related to a local database will grow linearly and may not be ignorable. For example, we can estimate that the number of data entries in each local database will be in the order of 10^6, which is a considerable amount, when the required time for searching in a local database becomes almost same as the one for finding a responsible agent at $N = 128$.

Then, how does the performance related to the shared database change in case that the total scale of N-TAP gets larger? Before considering this, we first investigate the nature of the shared database as a distributed database.

To consult how data entries are distributed among agents, we calculated the variation of a fairness index defined as follows on the last experiment:

$$F = 1 - \frac{N}{N-1} \sum_{i=1}^{N} \left(\frac{1}{N} - p_i \right)^2 \tag{3.1}$$

In this equation, N denotes the number of N-TAP agents and p_i $\left(0 \leq p_i \leq 1, \sum p_i = 1\right)$ is the percentage of database entries hosted by the i-th agent ($i = 1, 2, ..., N$). F takes a value in [0, 1] and the larger value of F means the better fairness from the point of view that data entries are homogeneously distributed. The result is shown in Figure 3.13, whose X-axis shows the number of the finished trials of storing a data entry (or we can regard it as a time series) and the Y-axis is the value of F. After 45 trials, F becomes larger than 0.99 and keeps its growing trend until the 1000th trial finishes in all cases. This means that collected data entries were distributed homogeneously among agents and all agents were almost equally burdened with storage cost. In N-TAP, a data entry is stored in the local database of an agent that is a collector and also stored in the local database of other agents that are responsible in the N-TAP's ID space as described in Section 3.1.5. Therefore, if a specific kind of data such as the RTT between certain two nodes is collected more frequently than other data, more data entries are stored in the local databases of certain agents and the fairness index is expected to be lower.

Now we come back to the question: "How does the performance of the shared database change if the scale of N-TAP grows?" Suppose the number of agents becomes twice. Then we can expect that turn around time for finding a responsible agent increases in $O\left(\log_2 N\right)$ as with Figure 3.10. If an application that uses N-TAP is interested in obtaining network characteristics information of certain targets, now we can consider that the application has obtained N new measurement points and the number of measurement targets also becomes twice. In case that an application is interested in obtaining network characteristics information among agents (e. g., the IP-level topology among overlay nodes), the number of measurement targets rises in $O\left(N^2\right)$, so the number becomes about 4 times. Coincidentally the twice number of agents means that the fraction of data entries that each agent has to keep in its local database becomes the half. Hence the actual number of data entries that an agent has in a local database is expected to be the constant order in the former case and grow twice in the latter case. Looking back at Figure 3.11, the required time for handling a local database is almost constant or proportional to the database size. Namely, when the scale of the N-TAP system grows, we can expect that the required time for finding a responsible agent will still remain dominant in the former case. And the time for searching in a local database will overtake the one for finding a responsible agent in the latter case if the size of a local database becomes quite large.

Lastly, we refer to another feature realized on the N-TAP's measurement overlay network: communication channels among N-TAP agents. To establish a communication channel with another agent, the N-TAP agent has to search the data entry of the information (e. g., IP address and port number) on the target agent and then start communication. Therefore the required time just before the agent starts communication is same as the time for searching a data entry in the shared database. In case that an agent uses a connection-oriented protocol such as TCP to communicate with other agents, the required time for establishing a communication channel will include the communication delay between agents, which depends on their network proximity and the state of agents themselves.

3.3 Performance Evaluation of the overall N-TAP Network

3.3.1 Experiment

For this experiment, we used 128 homogeneous nodes in StarBED [42], which is a large-scale network experiment facility. Each node had an Intel Pentium III 1 GHz CPU, 512 MB memory and a 30 GB ATA hard drive. These nodes were connected through 100 Mbps Ethernet links in the same network. The Debian GNU/Linux operating system with the 2.6-series kernel was installed on each node.

We had one N-TAP agent run on each node; therefore we constructed a measurement network with 128 agents (i. e., $N = 128$). An N-TAP ID, which puts an agent in the Chord ring, was randomly assigned to each agent with no overlaps. The reason we chose random IDs was to distribute the load derived from maintaining the N-TAP network among the core agents in the hybrid and pure peer-to-peer measurement networks. After the N-TAP network was constructed, we ran a client program on one node that is in the same experimental network and did not have an agent. The program sequentially issued 2000 requests to one of the core agents for the RTT information between two randomly chosen experimental nodes. The program also issued the same number of the requests to one of the stub agents if the N-TAP network had stub agents. The request messages were exchanged based on the XML-RPC protocol between an agent and the client program. We note that an N-TAP agent usually tries to reuse RTT data previously collected and stored in the shared database if a client program specifies the request on the freshness of the RTT data. However, for simplicity in this experiment, we forced the agents not to reuse the RTT data but to perform the actual measurement. The agents logged their operations with time stamps, and N-TAP related packets were captured on the nodes, so we were able to analyze the behavior of the measurement network. We selected the values of 1, 2, 4, 8, 16, 32, 64 and 128 for C (the number of core agents) to shift a measurement network from the centralized one to the decentralized one. For convenience, we numbered the respective agents from 1 to 128 according to the following rules: (a) The first agent was a core agent that accepted and processed the above requests. (b) If there were other core agents, they were numbered from 2 to C. (c) If there were one or more stub agents, we set a stub agent that accepted and processed the above requests as the 128th agent. (d) If there were other stub agents, they were numbered from $C + 1$ to 127. Also note that the 128th stub agent was configured to issue a request related to the N-TAP network to the first core agent.

The procedures carried out by a core agent when it accepted an RTT measurement request are as follows:

1. The core agent searches the source node in the requested RTT measurement. In this procedure, the core agent issues a request to find a core agent that is responsible for storing the data entries on the source node in the shared database. After it finds a responsible agent, it asks the agent to send the information on the source node (for instance, whether the source node is alive or not).

2. If the source node is alive (this condition is always true in this experiment), the core agent asks the source node to measure the RTT. Then the source node sends the measurement result to the core agent.

3. On receiving the result, the core agent responds to a client program with this result.

4. The core agent stores the collected RTT information in the shared database. It finds another core agent, one that is responsible for storing this data entry, and sends the entry to the responsible agent.

In the case of a stub agent, a control message to find a responsible agent was always sent to a specific core agent because the stub agent did not have a routing table in the N-TAP network but only knew the core agent that bridged between the N-TAP network and the stub agent itself. Apart from this messaging manner, the stub agent behaved in a same way as a core agent.

After the experiment, we confirmed that no measurement error had occurred and that all of the N-TAP related packets had been correctly captured during the experiment. The evaluation carried out in the following section is based on the recorded behavior of the agents after the measurement network became stable, i. e., no change in the agents' routing tables were made.

3.3.2 Evaluation

In this section, we investigate the basic characteristics of the respective measurement networks shown in Section 3.1.4. Our focus is the load distribution and the responsiveness to a measurement request in measurement networks.

Load Distribution

First we investigate the flow of control messages in the N-TAP network. Since the N-TAP agents have to carry out procedures according to the control messages, we can determine the distribution of loads among the agents by seeing this flow. Figure 3.14 depicts the distribution of exchanged control messages among the agents. Its horizontal axis denotes the assigned numbers of source agents in control messages, and the vertical axis denotes the assigned numbers of destination agents. The colored squares in the graphs show the number of the messages by their darkness: dark gray indicates more messages were exchanged and light gray means fewer. Specifically, where we define M as the logarithm of the number of exchanged messages, we divide the zone of the values of M into four even intervals and assign four shades of gray to the respective intervals so that the zone of the largest value of M is the darkest; a white area means that no message was exchanged between the agents. The horizontal and vertical dotted lines indicate the borders between the core agents and stub agents; therefore the bottom-left area shows the messages exchanged between two core agents, the bottom-right and top-left areas are for the messages between a core agent and a stub agent, and the top-right area is for the messages between two stub agents.

In any case, we can confirm that the squares are plotted more densely in the bottom-left area than in other areas, and the grays there are mostly dark. This shows that the burden of maintaining the measurement network was concentrated on the core agents, and the stub agents were relatively freed from such tasks. Additionally, no message was exchanged between two stub agents except for the cases of involving the 128th agent. The reason why the number of the messages to/from the first and 128th agents is large is that these agents had to ask other agents to perform the RTT measurement when they accepted measurement requests. For example, these agents asked the 10th agent to obtain the RTT between the 10th agent and the other agents. Moreover, after they obtained the RTT information, the first and the 128th agents had to store the measured RTT information in the shared database, as described in Section 3.3.1.

Secondly, we look into the exact number of exchanged messages and its tendency. Figure 3.15 shows the total number of exchanged messages during the 4000 requests in the respective cases of the C values. We can find that the number of messages exchanged between two core agents increases proportionally as the logarithm of the number of core agents grows. This number is zero where $C = 1$ because there is only one core agent and it does not need to issue a control message to another core agent. Meanwhile, the number of messages exchanged between a core agent and a stub agent changes slightly, though it becomes zero in the case of no stub agent ($C = 128$). The number of messages exchanged between two stub agents decreases as the number of

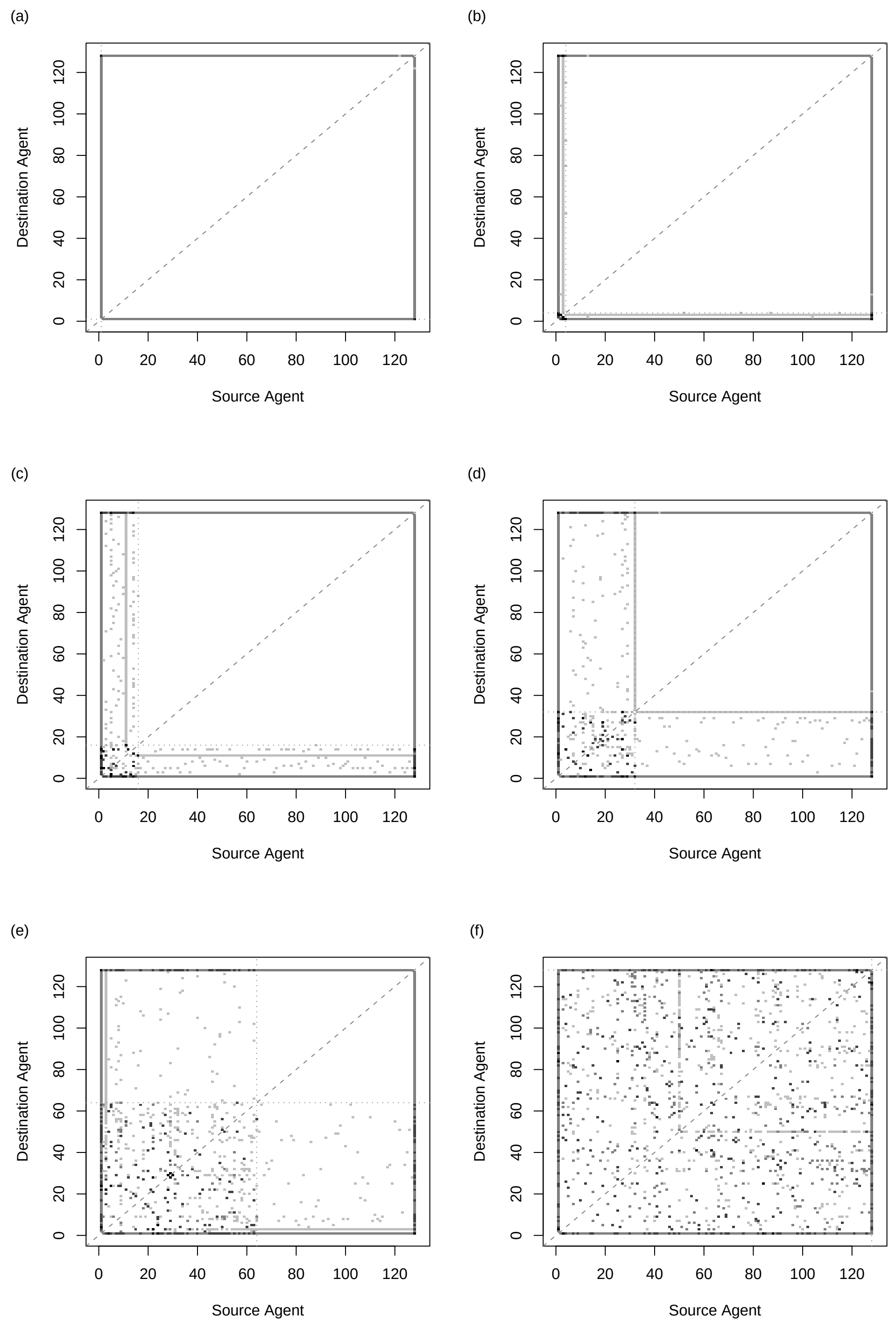


Figure 3.14: Distribution of exchanged messages among 128 N-TAP agents where (a) $C = 1$, (b) $C = 4$, (c) $C = 16$, (d) $C = 32$, (e) $C = 64$, and (f) $C = 128$.

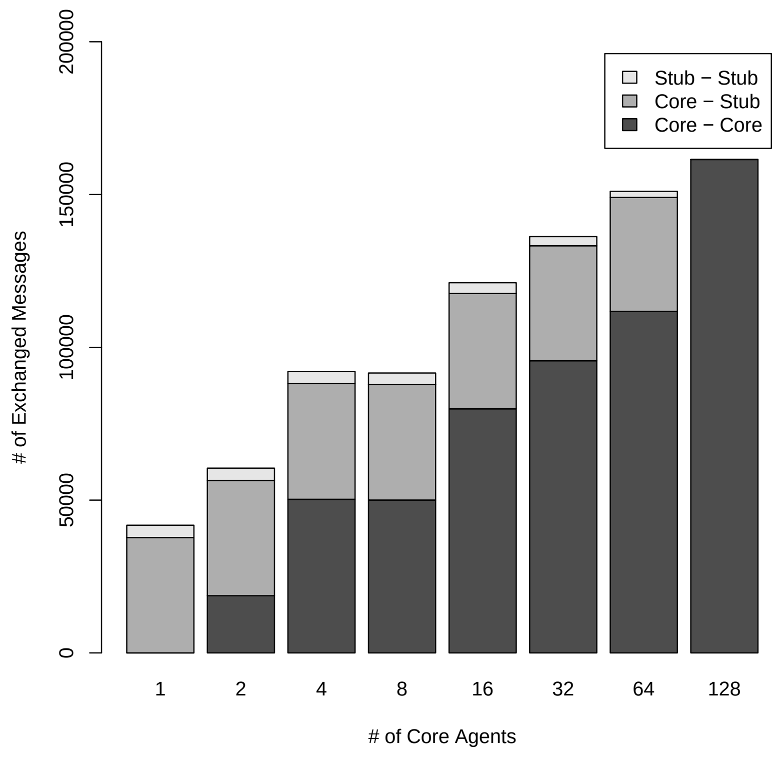


Figure 3.15: Number of exchanged messages.

stub agents decreases. The total number of messages tends to be larger as the number of core agents increases.

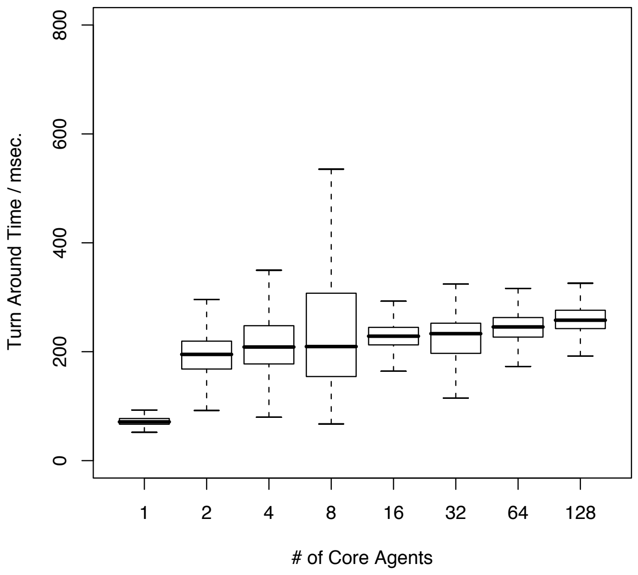
From these tendencies, let us see the number of messages that an agent of each role has to process as a metric of loads. It appears that the number of messages that one core agent has to process is reduced when the proportion of core agents to the total number of agents is large, because the growth order of the summation of the core-core and core-stub messages is lower than that of the number of core agents. On the other hand, when this proportion of core agents is large, the number of messages that one stub agent has to process increases but is still smaller than the number of messages that one core agent has to process.

These facts indicate that the scheme of core and stub agents works just as we had intended, that is, that the loads should be distributed among the core agents, and the stub agents should have less burden. The maintainer of the measurement network can easily adjust the load distribution with the proportion of core agents as he or she intends.

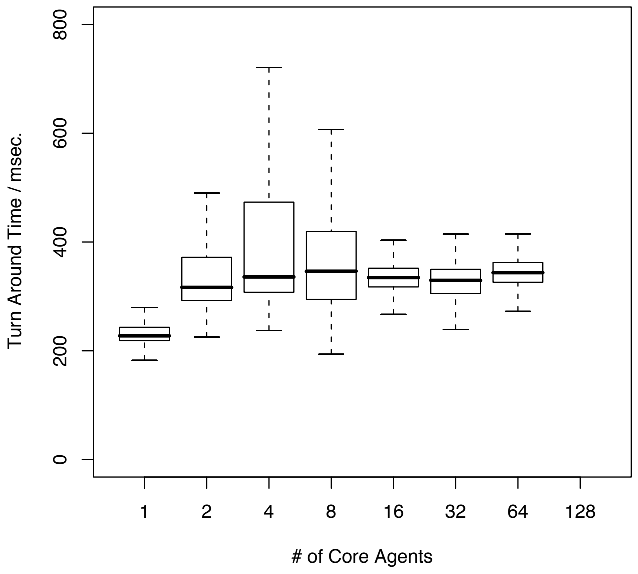
Responsiveness

Next we compare the responsiveness to a measurement request in the cases of a request to a core agent and to a stub agent. Responsiveness is an important factor as an application-oriented measurement service, because it has an effect on optimization procedures performed by autonomic applications that need network characteristics information. In Figures 3.16(a) and 3.16(b), the boxplots that represent the distribution of the turn-around time for a measurement request are depicted. Figure 3.16(a) represents the turn-around time in the case that a client program issued requests to a core agent, and Figure 3.16(b) represents the turn-around time in the case of issuing requests to a stub agent. In both graphs, the horizontal axis denotes the number of core agents and the vertical axis denotes the turn-around time for one request. In the case of sending a request to a stub agent, the boxplot where $C = 128$ is not given because we have no stub agent in the measurement network.

We can find that, in both cases, the turn-around time where we adopted the centralized model is shorter than the turn-around time with other models. The difference between the centralized model and other models is that the agent that accepts a request must perform the procedures for finding a responsible agent, retrieving the information on agents in the N-TAP network from the shared database, and requesting other core agents to store the collected data as an entry in the shared database. Inspecting the log files of the agents that accepted measurement requests from a client program, we calculated the mean of required time for each procedure, and the result is shown in Table 3.1. From this table, we see that, between the centralized model and the other models, a considerable difference of the time required for a measurement request is dominated by the time required for these procedures. The time required for finding a responsible agent is expected to increase linearly depending on the logarithm of the number of core agents. This is because, given the nature of Chord, the number of times a control message to find a responsible agent is forwarded among the core agents is proportional to the logarithm of the number of core agents. In Table 3.1, the required time for finding a responsible agent seems to follow this expectation. On the other hand, the required time for the database-related procedures would not significantly change while the size of local databases in respective core agents is small, and we can confirm such a tendency from the table. We also note that the distribution of the core agents' IDs also has an effect on the topology of the measurement overlay network, which results in the fluctuation of the required time, as in the above table. In this experiment, the IDs were randomly



(a) Request to a core agent



(b) Request to a stub agent

Figure 3.16: Turn-around time for a measurement request.

Table 3.1: Required time for the procedures (in milliseconds).

The number of core agents ($= C$)	2	4	8	16	32	64	128
Find a responsible agent (core)	1.0	2.5	6.2	4.8	6.4	7.5	7.8
Retrieve from the shared DB (core)	5.5	5.3	7.6	3.5	3.4	3.0	3.0
Store in the shared DB (core)	52.7	53.7	49.4	58.2	55.5	60.5	62.6
Find a responsible agent (stub)	2.1	7.1	9.6	8.3	9.8	11.2	—
Retrieve from the shared DB (stub)	10.0	9.1	9.7	5.5	5.2	4.6	—
Store in the shared DB (stub)	56.5	58.5	55.0	60.3	56.4	61.5	—

assigned; therefore we suppose that the required time for these procedures is almost the same among the agents.

The required time for finding a responsible agent in the case of sending a request to a stub agent is longer in the order of a few milliseconds than in the case of sending a request to a core agent. This can be explained by considering that a stub agent first needs to send a control message to a core agent, while a core agent can send a message directly to the next hop's agent in its own routing table. We can suppose that this additional procedure for a stub agent increases the required time in the case of sending a request to a stub agent.

According to the discussion in this section, a measurement network with the centralized model is superior to one with the hybrid or the pure peer-to-peer model in its responsiveness to a measurement request. In this experiment, communication delay between the agents is short enough to be ignored, however, the communication delay will range approximately from tens to thousands of milliseconds when the measurement network is deployed in a wide-area network. This will have significant influence on the measurement network with the hybrid or the pure peer-to-peer model because a larger number of control messages must be exchanged through networks in these measurement networks. However, the centralized measurement network always has to struggle with load concentration at a core agent. These factors should be considered in constructing a measurement network.

3.4 Discussion

First, based on the performance evaluation in Section 3.2, we discuss the tactics for the rapid provision of network characteristics information to applications. To quicken the collection of network characteristics information, we introduced the "local-first and remote-last" rule in Section 3.1.5. We can summarize four procedures taken in this rule as follows: Searching in a local database can be done relatively rapidly in most cases and does not generate extra network traffic. Required time for the actual measurement of network characteristics information depends on respective measurement methodologies. Both searching the data collected by other agents and requesting measurement to another agent involve the search in the shared database, therefore the required time for these respective procedures are estimated to be longer than the one for searching in a local database. Additionally, the procedure of requesting measurement invokes measurement activity on another agent, thus, it must pay more time cost than just searching in the shared database. Standing on these prospects, we can confirm that the "local-first and remote-last" rule is a reasonable tactic in point of its time-cost based prioritization of these four

Table 3.2: Considerable cases on utilizing N-TAP with the "local-first and remote-last" rule.

LD	MS	SD	RE	[P]ros and [C]ons
√				[P] Rapid provision of network characteristics information [P] No measurement overhead
√	√			[P] No need for performing measurement by an application itself [C] Slight time-cost penalty
√	*	√		[P] Provision of network characteristics information that cannot be collected by a solo node [C] Time-cost penalty
√	*	√	√	[P] Provision of network characteristic information that cannot be collected by a solo node [C] More time-cost penalty

LD: Search in a local database
MS: Measure by an agent itself
SD: Search in the shared database
RE: Request measurement to other nodes

procedures.

Looking from the viewpoint of applications, in Table 3.2, we show the pros and cons of utilizing N-TAP with the "local-first and remote-last" rule compared to performing measurement procedures by applications themselves. The tick mark in this table means that the marked procedure is done in the rule, i. e., the row with the marks of LD and MS means the case that an agent first tried to search in a local database but failed so it performed measurement and then provides collected data to applications. The asterisk shows that the corresponding procedure may not be invoked in the sequence of procedures. In case that the "local-first and remote last" rule ends at the point of searching in a local database, applications can receive desired network characteristics information rapidly. In the case of ending at the point of performing measurement, applications can retrieve network characteristics information without actually performing measurement by themselves but with a slight penalty of the failure of searching in a local database. If the data exist in the shared database, applications can retrieve network characteristics information that cannot be collected by a solo node with paying some time costs. In the case of requesting measurement to other agents, applications also have a chance to obtain such kind of network characteristics information, but they have to pay more time costs to complete this procedure.

As an idea for modifying the "local-first and remote-last" rule, we consider to reverse the turn of searching in the shared database and requesting measurement to other agents. If the measurement procedure can be done quickly and succeed in a higher rate than the one of finding desired data entries from the shared database, this modification will count. Anyway since the effect of such modification depends on the characteristics of respective measurement procedures, we have to carefully investigate the characteristics and make tactics for performing measurement in the future.

In the latter evaluation section, we described the trade-offs among measurement networks with three different models based on the agents' behavior in respective networks. The centralized measurement network can get the best responsiveness in exchange for the heavy loads, which may bring a decrease in responsiveness. In the hybrid measurement network, we can select multiple core agents according to our purposes, and the processing loads can be distributed among the core agents. The load on one core agent will be the minimum on an average in the case of the pure peer-to-peer measurement network. However, in the hybrid and pure peer-to-peer measurement networks, the responsiveness will go down depending on the size of the control planes of these networks.

The ease of adjusting the formation of a measurement network will be important in the actual deployment of a measurement service. In this paper, we first proposed the scheme of core and stub agents in a measurement network. With this scheme, we can easily shift the measurement network among the centralized network, the hybrid network and the pure peer-to-peer network by adjusting the proportion of core and stub agents. In the case that we can control a measurement network (e. g., when we monitor network facilities with such measurement systems), administrators should design the measurement network to meet their requirements. They will benefit from the ease of adjustment to the measurement network. In the case that we cannot know beforehand what types of agents will join a measurement network, we cannot create a clear plan for constructing the network. One of the cases is that the agents run on the same nodes as the applications (an overlay network application, etc.), whose nodes will arbitrarily join and leave. Even in such cases, role-based adjustment will work with the application nodes. For example, in order to improve the responsiveness to a measurement request, we would choose agents that are connected with a high-speed link and have high performance as core agents. Other metrics, like the continuous running time of nodes, will also be helpful in constructing the desired measurement network.

Focusing on the application-oriented measurement service, quick responsiveness to a measurement request is indispensable in a measurement system. To improve the responsiveness in a hybrid or a pure peer-to-peer measurement network, some possible refinements of a measurement system can be pointed out. One is to let an agent cache the results of finding a responsible agent so as to decrease the number of exchanged control messages. From the results in Section 3.3.2, in a large-scale core network, we can expect that the required time for finding a responsible agent will become dominant in the turn-around time for a measurement request. Caching the results of this procedure will improve the responsiveness, but the agents will need to handle the inconsistency between the cache and the actual topology of a measurement network, and we will pay a waiting time penalty when such inconsistency occurs. Moreover, as described before, choosing core agents based on the capability of agents will also be effective. In the case of choosing core agents dynamically, we will also have to handle the migration of key-value pairs in a distributed hash table (DHT), which is expected to be a considerable burden.

In a hybrid peer-to-peer network, each overlay node is assigned one or more node roles and is managed in a hierarchical structure as described already in this paper. Kazaa [38, 43], a peer-to-peer file sharing application, utilizes this scheme to connect between its unstructured peer-to-peer network and ordinary nodes. The extension to N-TAP that we have added in this chapter is unique in applying this scheme to a structured measurement overlay network in which measurement procedures different from the ones of ordinary file sharing applications are performed.

3.5 Related Work

In this section, we first introduce some cooperative measurement methodologies that are expected to be implemented on N-TAP and other application-oriented measurement platforms.

Additionally, we also present existing measurement platforms and their positions against N-TAP.

3.5.1 Cooperative Measurement

The idea of cooperative measurement is simple. In cooperative measurement, measurement nodes have same objectives of their measurement activities such as accelerating the speed of measurement, extending the coverage of measurement, and estimating network characteristics information without individual measurements to reduce measurement traffic. To achieve their objectives, they perform some cooperative actions that involve other measurement nodes according to their common rules.

A typical example of cooperative measurement is the network proximity estimation based on a coordinate system on which Internet nodes are plotted to obtain network characteristics information. For instance, Vivaldi [15] is a decentralized coordinate system that enables the estimation of the RTT between two nodes while performing fewer measurements. In Vivaldi, each node refers the coordinates of other nodes and calibrates its own coordinate according to a physical spring-mass system. The estimated RTT between two nodes is calculated by the Euclidean distance between them in the coordinate system. GNP [44], NPS [45], Lighthouse [46], PIC [47] and Netvigator [48] are also included in the systems that estimates network characteristics information based on geometric information.

To reduce measurement overhead for IP-level topology discovery, Donnet et al. introduced the Doubletree [12] algorithm. They noticed the fact that the respective results of topology probing include a high number of overlapping routes and redundant probes cause extra measurement overhead. To address this problem, the measurement nodes of Doubletree exchange their measured topology data and extract the common portions among measured topology data. Based on such analysis, they decide which parts of the topologies do not need to be visited repeatedly. In addition, Doubletree uses the technique of Bloom filters to reduce the bandwidth consumption derived from the communication among measurement nodes.

We can pick out two patterns of cooperation from these methodologies: one is that their measurement nodes communicate with other nodes for controlling their measurement procedures, and the other is that the measurement nodes share their collected data. For example, in Vivaldi, each node needs to communicate with other nodes to know their coordinates. Another example is that the Doubletree nodes share collected topology data among them. Therefore we regard these two features the ″communication channels″ and the ″shared database″ as the essential features for implementing cooperative measurement methodologies.

3.5.2 Measurement Platforms

There are a number of the projects for collecting network characteristics information on their large-scale measurement platforms. CAIDA [37] is one of the largest group performing the Internet measurement and analysis. Its projects cover the collection and analysis of various kinds of network characteristics information including IP-level and AS-level topology and the performance of network services. Additionally they study on their measurement methodologies. DIMES [49, 50] and NETI@home [51, 52] are also the famous projects of distributed measurement. Their concepts resemble the one of a distributed computing project called SETI@Home [53, 54]; the measurement nodes send measured data to a central server and the server performs the analysis for the study of the Internet structure. The measurement platforms of these projects are mainly designed with the focus on scientific and statistical analysis rather than the utilization of network characteristics information by applications. So their objectives are different from the one of N-TAP, however, the fundamentals of their studies, such as the deployment manner of infrastructure and measurement techniques, are also informative for N-TAP.

Pandora [7] is a well-constructed programming platform for autonomic applications. Pandora provides the programming components that simplify the procedures of network monitoring and packet processing. By providing these components, Pandora approaches to its goal of encouraging applications to obtain and utilize network characteristics information. On the other hand, N-TAP takes an approach of wrapping up the procedures of measurement methodologies and letting them available as an independent service.

Some systems such as pMeasure [55], iPlane [56] and S^3 [57] have almost same goals as N-TAP from the viewpoint of the provision of network characteristics information to applications. These systems are deployed as distributed measurement services, which collect and provide network characteristics information. While they mainly focus on measurement methodologies that are available on them, the investigation of basic requirements for the architecture of an application-oriented measurement platform is still left. Such requirements definition and the evaluation based on the requirements are also necessary along with the deployment of measurement methodologies on the platform. On the architectural aspect, N-TAP differs from these systems especially in the manner of measurement activity. Each N-TAP agent stores collected data in both local database and the shared database, and according to the "local-first and remote-last" rule, the agents prioritize the search in their local databases and the measurement on a local node in order to improve response time and reduce measurement overhead.

3.6 Summary

In this chapter, we presented the architecture of a distributed measurement platform named "N-TAP" that works in a peer-to-peer manner. N-TAP acts as a measurement service that provides network characteristics information for autonomic applications like overlay network applications, and enables the implementation of cooperative measurement methodologies on it. Through the performance evaluation of fundamental features that are tested in an actual network environment, we explored the effects of each function on turn around time. From the viewpoint that the rapid provision of network characteristics information is important for autonomic applications that utilize such information, we also discussed the tactics for measurement activities so that these applications can rapidly retrieve network characteristics information from such kind of measurement platforms. Our investigation of the fundamental features of a peer-to-peer measurement platform from the aspect of time costs showed basic data to make tactics for performing measurement procedures on N-TAP, and we confirmed the "local-first and remote-last" tactic has certain reasonability from this aspect.

We also focused on a methodology for constructing a measurement network, which can easily change its network formation, alternating between centralized, hybrid and pure peer-to-peer models. By adopting this scheme and modifying an existing measurement agent, we investigated the operational flow in each of the measurement networks. As a result, we were able to confirm that exchanging control messages through networks has an appreciable effect on the turn-around time for a measurement request in the hybrid and pure peer-to-peer measurement networks. At the same time, the processing loads were successfully distributed among core agents in these networks. The consideration of such trade-offs is important in constructing a desired measurement network.

Chapter 4

Unleashing a Cooperative Measurement Methodology on an Actual Network

Even though we have a number of sophisticated cooperative measurement methodologies, they have not yet appeared as a core measurement technique on actual monitoring systems. In this chapter, to validate that a cooperative measurement methodology can be implemented on N-TAP and it can bring advantages for the Internet observation, we build an actual topology probing system based on one of the cooperative measurement methodologies, called Doubletree, which discovers IP topologies in the distributed manner. 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. We describe our implementation of distributed topology measurement (DTM) system, 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). Through the deployment of DTS on the PlanetLab testbed, we demonstrate that DTS presents strong advantages in terms of flexibility, robustness, scalability and modularity.

4.1 Background

The research work on the Internet topology has been emphasized for these years. The work is based on the topology maps built by systems such as skitter [58], which probes the Internet topology from multiple vantage points using the technique of traceroute [59]. We call these distributed topology measurement (DTM) systems.

A milestone result from DTM data was Faloutsos et al.'s paper [60] on power-law relationships in the Internet. They found that the distribution of router degrees follows a power law. Their map came from an early DTM, by Pansiot and Grad [61], consisting of just twelve traceroute hosts. The solidity of a result based upon so few vantage points was put into question by Lakhina et al.'s finding [62] that, in simulations of a network in which the degree distribution does not at all follow a power law, traceroutes conducted from a small number of monitors can nonetheless induce a subgraph in which the node degree distribution does follow a power law. Clauset and Moore [63] have since demonstrated analytically that such a phenomenon is to be expected for the specific case where the actual topology is an Erdös-Rényi random graph [64], which is far from a power-law graph. In order to remove potential spatial bias from the Internet maps, therefore, we require much larger scale DTMs even though we are yet to know precisely how many monitors are needed.

Another motivation for deploying a large-scale DTM is to better track 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.

One rapid way to deploy a large scale DTM would be to deploy traceroute monitors in an easily downloadable and readily usable piece of software, such as a screen saver. This was first proposed by Jörg Nonnenmacher, as reported by Cheswick et al. [65]. The approach is inspired by distributed computing tools like SETI@home [53, 54]. The first publicly downloadable DTM is DIMES [49, 50], released in September 2004 as a daemon program that performs the monitoring on each end node.

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 [12, 66]. The NSF-sponsored CONMI Workshop [67] 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.

Topology discovery can be enforced through a centralized or decentralized architecture. Current systems, such as Scriptroute [68], Rocketfuel [69] and DIMES [49], are centralized. Unfortunately, such an architecture brings with it necessary scalability problems and a single point of failure. A centralized system might suffer strain on its server and network link, given a large enough number of participants and control messages. Further, if the centralized communication entity fails, the entire system fails.

In this chapter, we propose the first decentralized architecture for a DTM. We build on our prior work [12] in introducing cooperation among DTM monitors, through the Doubletree topology discovery algorithm. Doubletree takes advantage of the tree-like structure of 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. In other words, the key idea of Doubletree is that the 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. 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. By adopting the decentralized architecture, we can expect the problems stated above are addressed or moderated. At the same time, we may face a problem that the decentralized architecture has. Therefore, we should investigate the merits and demerits of these architectures stepwise.

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. These features are found in distributed hash tables (DHTs). One DTM that uses a DHT architecture for just this reason is N-TAP, which stores its data in a DHT-based distributed database.

Our proposed architecture is called Decentralized Tracing System (DTS), which makes use of DHTs for decentralizing the control and data planes of a DTM. The control plane concerns how to guide a monitor's probing while the data plane refers to collected data storage. In order to decentralize both planes, DTS considers three DHTs, each one being dedicated to a particular type of information that must be shared between monitors: the probing control information (i. e., indications about paths already explored by monitors), the probing target (i. e., the list of destinations to probe), and the probing data (i. e., the topological data collected).

We implement and deploy DTS on the PlanetLab testbed. Through this deployment, we demonstrate the merits of DTS with investigating the characteristics of the Doubletree algorithm implemented on it. We mainly show that DTS is flexible (i. e., each monitor can probe at its own pace without disadvantaging the rest of the system), robust (i. e., DTS continues to work in case of one or several monitors fail), scalable (i. e., DTS scales against the growth of the number of monitors), and modular (i. e., DTS is designed in a modular fashion, which divides a program into several independent components). Finally, DTS is open-source and freely available to the research community under a BSD-like license.

4.2 DTM Systems Requirements

In this section, we discuss the requirements for any DTM system. One can see a DTM system being composed of two planes: the control plane and the data plane. The control plane, detailed in Sec. 4.2.1, concerns how to guide a monitor's probing. The data plane, discussed in Sec. 4.2.2, refers to the collected data storage.

4.2.1 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 chapter, we refer to the target list as probing target.

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.

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. Such a dynamic behavior must have limited impact on the shared information. Robustness and flexibility must also be key properties of a DTM system.

Finally, the control plane of a DTM system must ensure that each probing monitor can perform measurements at its own rhythm. A slower monitor cannot slowdown others monitors.

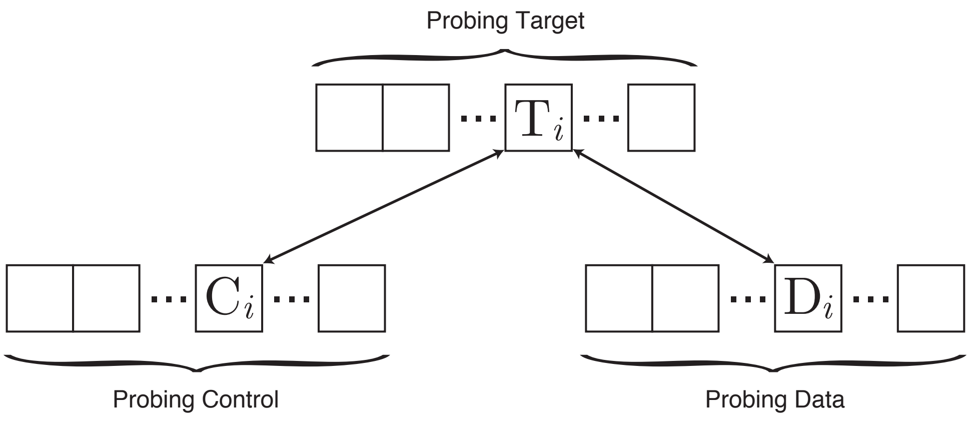


Figure 4.1: Relationship between information shared among DTS monitors.

4.2.2 Data Plane

The data plane of a DTM system refers to the topological data collected during probing. In the fashion of the skitter [58] data, this data set might be accessible by the research community. A DTM system has to keep track of each probing result, for each probing monitor, from the beginning and must ensure the long-term persistence of this data set.

Storing data collected during probing can lead to scaling issues. For instance, since January 1998, CAIDA stored around 3.343 TB of skitter data. This data set corresponds to 59,578 trace files containing a total of 12,271,674,523 traces. These traces have been produced by a set of monitors evolving from five to 24 towards a set of several thousands destinations. Finally, note that CAIDA keeps backup of this data, increasing therefore the storage volume needed. We expect that the storage volume will increase as we scale up the number of probing monitors.

The DTM system must provide an easy access to the data storage system. On one hand, probing monitors must be able to efficiently and easily store the data collected. 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.

4.3 Design and Implementation

In this section, we describe the Decentralized Tracing System (DTS), the first entirely distributed topology discovery system. We design and implement DTS so that it meets the requirements provided in Sec. 4.2. We first provide a high-level view of DTS (Sec. 4.3.1) and, afterward, we detail how the control and data planes have been implemented (Sec. 4.3.2 and 4.3.3 respectively). We finally describe N-TAP (Sec. 4.3.4), the measurement platform on the top of which DTS is built.

4.3.1 Global View of DTS

In Sec. 4.2, 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.

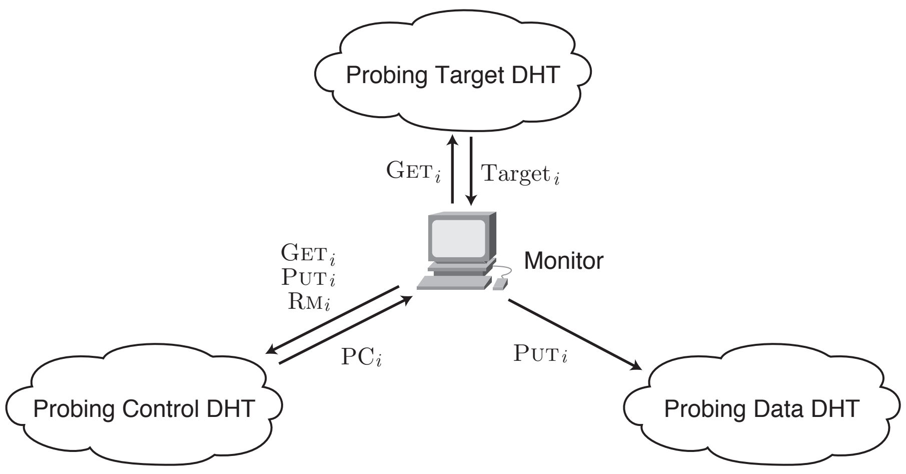


Figure 4.2: DTS and the dedicated DHTs.

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 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. 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. 4.1 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 distributed hash tables (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. 4.2.

Each value stores by a specific DHT refers to a chunk. For instance, the Probing Target DHT on Fig. 4.2 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.

The DHT labeled Probing Target DHT (PT DHT) is used to store the target list chunks. A monitor has a read-only access, in order to obtain a specific target chunk. A third party can interact with the PT DHT by adding (the Put primitive) or removing (the Rm primitive) IP addresses in a chunk. Removing occurs when a particular IP address must be black-listed as its owner does not want to receive probes.

The DHT labeled Probing Control DHT (PC DHT) is used to store probing control information. As DTS implements the Doubletree algorithm [12], a probing control information refers to a stop set. A stop set is a data structure that contains (interface, destination) pairs encountered during probing (see Sec. 4.3.2 for details). Each DTS monitor must be able to query a stop set for the existence of a given pair as well as to populate the stop set with pairs encountered. A stop set

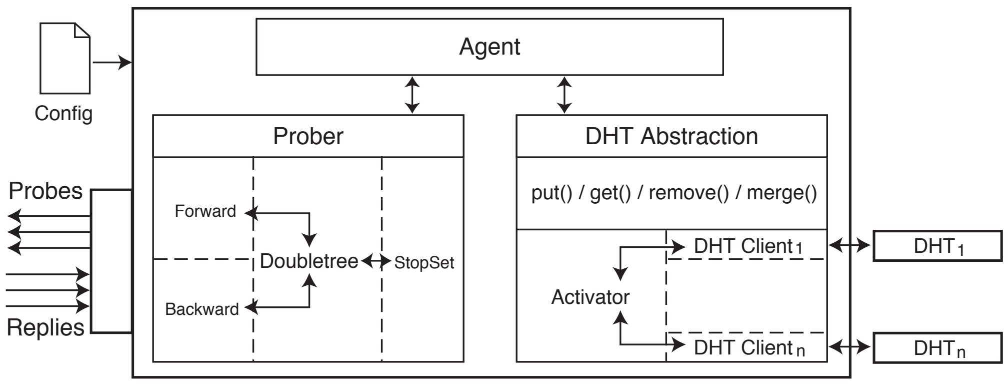


Figure 4.3: DTS' control plane.

is valid during a certain amount of time due to network dynamics, i. e., routing changes.

The DHT labeled Probing Data DHT (PD DHT) is used to store topological data gathered by a monitor after probing a specific target chunk. Currently, a monitor has a write-only access to the PD DHT, using the Put primitive. Instead of centralizing all the results, we choose to make sure that the resulting topological information is stored in a persistent manner, so that it will remain even if the monitor in question departs from the system. Further, it has been recently shown that using DHTs to store huge amount of data is efficient [70, 71]. However, we are leaving for future work the question of how one might want to organize and query a DHT that contains all the results. The current design allows a data analyst to request chunks of data by monitor, chunk of destinations, and rough time interval.

The advantages of the DTS infrastructure are the following:

- Flexibility: Monitors can join and leave the system at will. In addition, each monitor probes at its own pace. A slower monitor does not disturb the system by forcing others to adapt their rhythm.

- Robustness: If one or several monitors fail, the system continues to function, and relatively little work is lost. Actually, the loss is dependent on the chunk size. The larger the chunk, the higher the potential loss. Similarly, the smaller the chunk, the smaller the impact of a loss. However, smaller chunks might imply more interactions with DHTs which can lead to an increase in the global load generated by DTS. There is thus a trade-off to find between chunk size and robustness.

- Scalability: Due to the distributed storage of crucial data (i. e., the stop set), as the number of participants (i. e., the monitors) increases, the ability to exchange stop sets does not get constrained by a central server.

- Modularity: DTS is designed in a modular fashion, making future extensions easy. There is also an API for interacting with external elements such as the DHTs (see Sec. 4.3.2). It can reuse existing infrastructure, such as existing DHTs, PlanetLab, and N-TAP.

4.3.2 Control Plane

The control plane of DTS is composed of several modules that interact with each other through the Agent engine. Fig. 4.3 shows the control plane of DTS.

A DTS monitor starts by loading a configuration file. This file contains, among others, information about the DHT gateways as well as the number of chunks in the system. To each chunk is thus associated a number. This number will help a monitor to generate the key needed when interacting with the DHTs. Based on the number of chunks, a monitor generates a random order for browsing the various chunks. The control plane interacts with the PT DHT to obtain the probing target corresponding to a chunk.

A DTS monitor probes the network by using its Prober engine. The Prober engine implements the Doubletree algorithm [12], which aims at significantly reducing probing redundancy. It takes advantage of the tree-like structure of routes in the Internet. 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 – decreasing TTLs) and forwards (destination-rooted tree – increasing TTLs).

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 global stop set if a monitor receives a packet from the interface in reply to a probe sent towards the destination address. As the local stop set concerts the monitor-rooted tree, each monitor manages its own local stop set. On the contrary, the global stop set refers to the destination-rooted tree and must thus be shared among monitors. Therefore, the control plane of DTS interacts with the PC DHT in order to store and retrieve the global stop set corresponding to the current chunk.

A Doubletree monitor 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 the 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. Note that in the special case where there is no response at distance h, the distance is halved, and halved again until there is a reply, and probing continues forwards and backwards from that point. Interested readers might find a discussion the calibration of h in our previous work [12].

Our approach in constructing DTS is somewhat similar to Chawathe et al. [72] 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 participants. One of the key element we had in mind when designing DTS was its ease of deployment. We therefore choose to make DTS free from DHT specifications. Instead of associating to DTS specific DHTs, 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. These interfaces allows 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 entities in DTS.

4.3.3 Data Plane

Our implementation of the data plane is somewhat similar to the control plane (See Fig. 4.3). 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. The data collected is, then, sends through the DHT abstraction to the PD DHT.

For each traced target, in addition to IP addresses and RTT of intermediate hops, the following information is stored: timestamp, stopping reason for backwards and forwards probing, and the distance at which probing stops (backwards and forwards). We envisage four different stopping reasons: 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 set (local stop set for backwards and global stop set for forwards), as defined in Sec. 4.3.2). At last, a *normal* stopping means hitting the first hop (backwards) or the destination (forwards).

Finally, a global information is associated to each target chunk, mainly the name of the DTS monitor, the chunk identifier, a timestamp and the value of the parameter h (see Sec. 4.3.2).

4.3.4 Adaptation to N-TAP

According to the design presented so far, we describe how DTS is implemented on an existing measurement platform, N-TAP. Basically, the N-TAP platform consists of a number of N-TAP agents that are assumed to reside in multiple administrative domains. Besides the agents perform measurement, such as topology discovery and RTT measurement, the agents also play a role in forming an overlay network with the technique of Chord [19], one implementation of DHT. The overlay network is called the N-TAP network, on which N-TAP provides some high-level functions such as the shared database and communication channels among agents. In N-TAP, there are two roles of agents (see Chapter 3). One of the roles is called core, and the other is stub. The core agents have to maintain the peer-to-peer network for the DHT service, meanwhile, the stub agents join the network not directly but via a core agent. Therefore, the stub agents do not have a burden on maintaining the peer-to-peer network. Being similar to the relationship between a super node and an ordinary node in the Kazaa [38] network, these two kinds of agents form a bi-layered peer-to-peer network.

Fig. 4.4 depicts the implementation of DTS on the N-TAP platform. First, we prepare several nodes for core agents that can serve the shared database. The number of the core agents has an impact on the scalability of DTS, therefore, we should carefully choose the number. We discuss the scalability of DTS in Sec. 4.4.3. 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 (PC DHT, PT DHT, and PD DHT) via a core agent. Briefly, core agents work as a gateway of

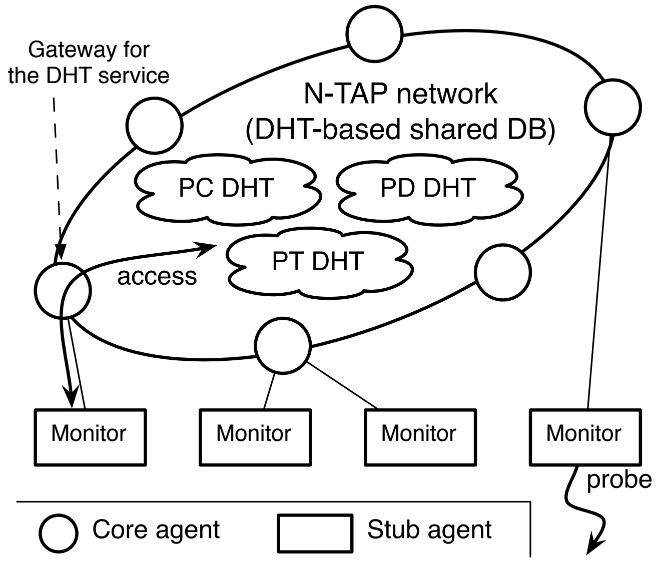


Figure 4.4: Implementation of DTS on N-TAP.

the DHT service for stub agents. Note that, for simplicity, we let DTS monitors be core agents in the experiments in the following sections only when the role of DTS monitors does not have an impact on experimental results. For preparing a list of probing targets shared in DTS, we make one agent commit the chunks of the target list into the PT DHT before monitors start topology discovery, or we let all monitors have the same list of targets by distributing a configuration file that contains the target list among the monitors.

The code of DTS is integrated into N-TAP and it works as one measurement module of N-TAP. After we configure N-TAP agents (DTS monitors) and give them essential information such as the number of chunks, we can start DTS by issuing the `dbltree start` command from the command-line interface of N-TAP. Then the monitors fetch one of the chunks randomly, and based on the Doubletree algorithm, they begin to perform topology discovery against the probing targets included in a received chunk. When a monitor finishes the topology discovery for one chunk, it fetches the next chunk and continue to probe. In the current implementation, a monitor stops probing when it finishes the topology discovery for all chunks, or we can let it keep running if we specify a option when starting DTS.

4.4 Evaluation

In this section, we perform the evaluation of our DTS according to the four indices presented in Sec. 4.3.1. We first discuss the flexibility of DTS (Sec. 4.4.1). We next proof the robustness of DTS (Sec. 4.4.2). Scalability and modularity are discussed in Sec. 4.4.3 and 4.4.4 respectively. Finally, Sec. 4.4.5 summarizes the important conclusion drawn in this section.

4.4.1 Flexibility

The flexibility of DTS is brought by the DTS architecture and the adoption of the Doubletree algorithm. Our architecture ensures a high degree of independence between monitors and the dedicated DHTs. This means that the behavior of monitors does not affect the information stored in these planes. Additionally, the nature of the Doubletree algorithm, that is, one monitor does not need to wait for measurement results or control messages from other monitors before probing but utilizes the results just for an efficient probing, is also inherited by DTS. In this section, we perform the validation of some merits of flexibility stated in Sec. 4.3.1.

Monitor's Joining and Leaving

One of the merits is that the DTS monitors can arbitrarily join and leave the system at will. This feature can be easily explained by considering the characteristics of DTS stated in the previous section. Moreover, a monitor decides its probing strategy by itself according to the Doubletree algorithm, and it is not forced any kind of operations by other entities. As shown in Fig. 4.2 and 4.4, a monitor indirectly communicates with other monitors through the shared information in the dedicated DHTs, however, it never communicates directly. Therefore, when a monitor joins the system, it just contributes to an efficient monitoring by committing entries of the global stop set. When a monitor leaves the system, the system loses one observation point, however, committed probing control information and probing data are not lost and the leaving does not cause a special operation to other monitors.

Impact of Slower Monitors

For similar reasons stated in the previous section, DTS has another merit that each monitor can keep its own pace for probing. Since the Internet consists of heterogeneous elements (e. g., nodes and links), we cannot expect that all monitors have enough resource and they can probe at their maximum paces; therefore, there must be a difference of probing pace among monitors. Even in such a situation, thanks to the characteristics of DTS that monitors are not forced any kind of operation by the behavior of other monitors, slower monitors do not need to catch up faster monitors, and faster monitors do not need to wait for the completion of probes by slower monitors. For instance, even if one monitor probes at a slower pace than other, the system and other monitors can work without considering the existing slower nodes.

From the aspect of flexibility, it is important to prove that even a slower monitor can contribute to the system. When the system has slower monitors, we can expect that the slower ones can make smaller contributions to efficient probing than faster monitors can do. More particularly, we suppose a slower monitor can averagely commit less entries of the global stop set than a faster monitor. However, we cannot easily estimate the fact on this matter quantitatively. We therefore conducted an experiment on the PlanetLab testbed to investigate this matter.

For the experiment, we randomly chose 16 PlanetLab sites and picked one node from each site, i. e. we selected 16 nodes that respectively reside in difference sites, and we deployed DTS on these nodes. Originally, the monitors are set to stub agents in DTS, however, for simplicity, the roles of all agents were set to core, which performs both probing and the management of the DHT service. This is why we do not need separate monitors and storage to investigate the contribution from respective monitors as long as they do not fail on the way. Probing targets were the monitors themselves, and the target list was shared in 16 chunks among the monitors, i. e., one target in one chunk. Then, the monitors start at the same time to probe the full mesh topology (16×16) among the monitors with the Doubletree algorithm. The monitors selected the destinations in a random order, therefore, the order of topology discovery differs in respective iterations. Each time a monitor discovers an interface, it commits a result to the global stop set, which is maintained in the PC DHT. The monitors logged the results of probing and the Doubletree-related actions, such as retrieving an entry of the global stop set for a specific destination or checking whether a discovered interface is contained in the stop set. The following results were analyzed from these logs. The monitors continued to probe until probes to all destinations were completed. From now, we call the monitors that probed at maximum pace "faster monitors," and the monitors that stopped for one second after each probe "slower monitors." Taking into account that one probe usually completes within tens to hundreds milliseconds, the pace of slower monitors is several times slower than the one of faster monitors. We prepared three sets of monitors for the experiment: In set 1, all the monitors were the faster monitors. In set 2, half of the monitors were faster ones, and the rest of the monitors were slower ones. In set 3, all the monitors were slower monitors. Monitors were distinguished by a unique ID from 1 to 16. The IDs were consistent through the experiment, and in set 2, monitors from ID 1 to 8 were faster ones, and 9 to 16 were slower ones. For each set, we iterated this topology discovery three times. For the following evaluation, we use the average values of respective indices in these three iterations.

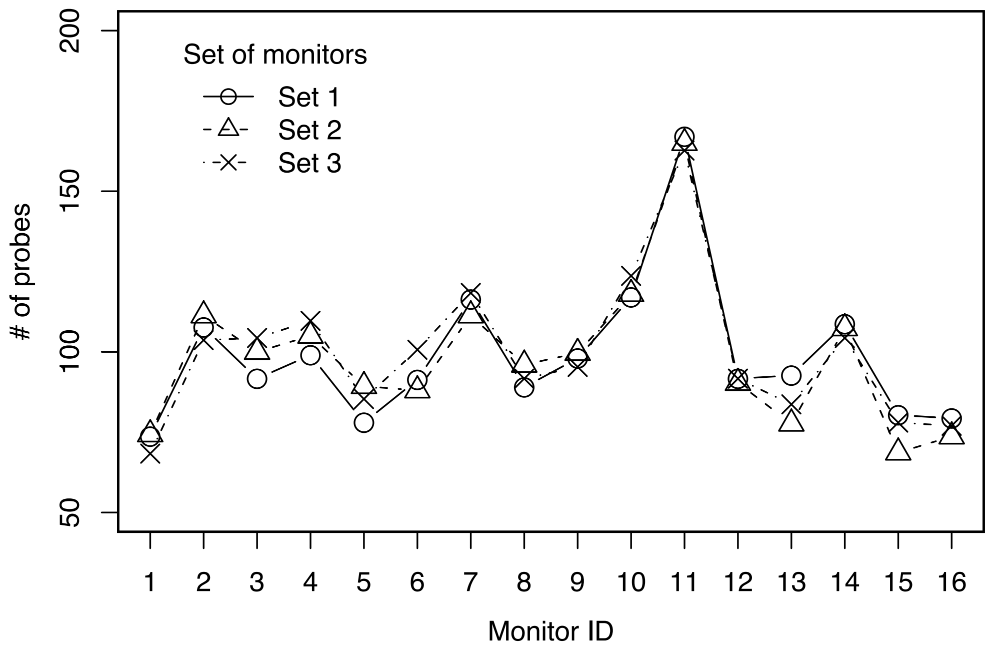
First, we look to the required time for completing probing and the number of performed probes. These values can be regarded as the indices of the behavior of faster and slower monitors. Fig. 4.5 shows the average time for probing and the number of probes in three iterations for each monitor. From Fig. 4.5(a), we can find out that both faster nodes and slower nodes had to probe almost same number of times even if the proportion of faster and slower monitors differed. In Fig. 4.5(b), we can see a tendency that faster nodes can complete within a certain time. However, time cost to slower nodes increases as the proportion of slower nodes grows. Considering these facts, we assume, in DTS, that both faster and slower monitors can find an entry of the stop set that can reduce redundant probes at an early stage and they can contribute to efficient probing even though slower monitors take more time to complete probing.

Secondly, to investigate how faster monitors and slower monitors respectively affect the overall efficiency of topology discovery, and to obtain more concrete reasons for the analysis above, we calculate a contribution index. We first define the contribution value with this equation:

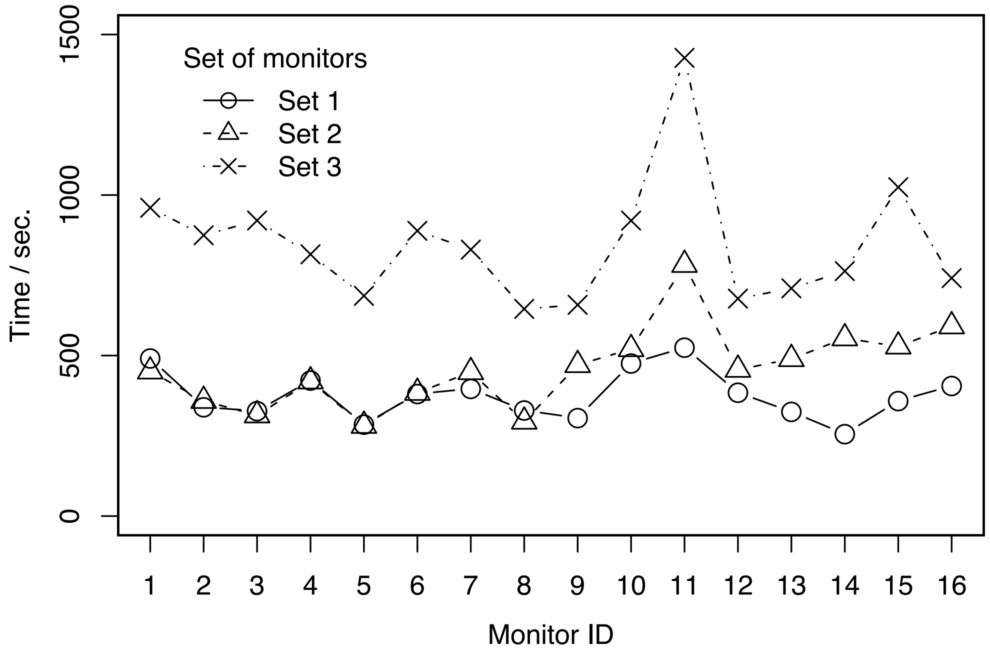
$$c_{g,i\rightarrow j} = \frac{U_{g,j} O_{g,i} H_g}{O_{g,f} + O_{g,s}}. \tag{4.1}$$

In this equation, g is an entry in the global stop set G, and $c_{g,i\rightarrow j}$ is a contribution value of g that means the contribution from i to j, where i and j are replaced with f (faster monitors) or s (slower monitors). In this context, the term "contribution" means how many probes that are expected to be performed by j are reduced by a stop set entry that i originated. H_g is the hop count between a discovered interface and a destination in g. With the technique of traceroute, a monitor can count the number of hops from the monitor itself to a target, however, cannot directly count the one from an intermediate node (interface) to a destination, e. g., the value of H_g. Here we calculated the value of H_g by subtracting the hop count between the monitor and the detected interface from the one between the source node and the destination, assuming that the route from the monitor to the destination did not change during topology discovery. This calculation can be expressed as the following equation:

$$\begin{aligned} H_g &= \text{Hop}\,(int, dst) \\ &= \text{Hop}\,(mon, dst) - \text{Hop}\,(mon, int) \end{aligned} \tag{4.2}$$



(a) Number of traceroute



(b) Required time

Figure 4.5: Average time for probing and the number of probing in the case of respective sets of monitors.

Table 4.1: Contribution index.

	$C_{f\to f}$	$C_{f\to s}$	$C_{s\to f}$	$C_{s\to s}$	Total	N_f	N_s
Set 1	1422.9	—	—	—	1422.9	1581.3	—
Set 2	617.4	203.3	102.9	499.0	1422.6	775.3	800.3
Set 3	—	—	—	1361.0	1361.0	—	1598.3

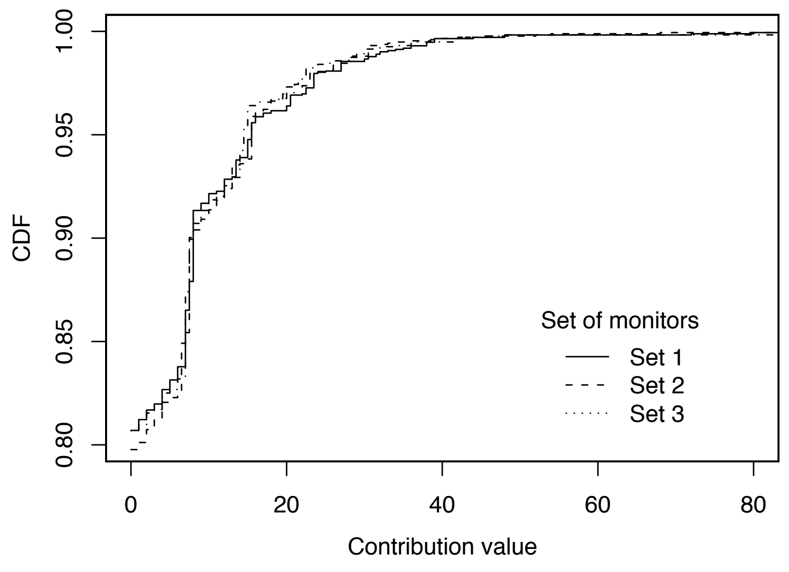
In this equation, Hop (p, q) denotes the number of hops from p to q, and *int*, *dst* and *mon* respectively mean a discovered interface, a destination and a monitor. Both numbers (Hop (mon, dst) and Hop (mon, int)) can be obtained through the topology discovery between the monitor to the destination. In the cases of some pairs of source and destination nodes, traceroute could not be completed because the Doubletree's halt( ) function returned `true`, that is, a loop was detected or a gap was discovered. In such case, we adopted the average of correctly calculated H_g values to the unknown H_g values. $O_{g,i}$ means the number of i that originated g, and $U_{g,i}$ is the number of i that utilized g. Depending the timing for inserting or retrieving entries of the global stop set, multiple monitors can originate same entries of the global stop set and insert them. Consequently, $O_{g,i}$ can be larger than 1, and $O_{g,f}$ and $O_{g,s}$ can be larger than 0 at the same time. Considering this fact, in the case that multiple monitors originate the same entry, we average the contribution value among these monitors by dividing it by the number of these monitors $(= O_{g,f} + O_{g,s})$. Finally, the contribution index is defined with the following equation:

$$C_{i\to j} = \sum_{g\in G} c_{g,i\to j}. \tag{4.3}$$

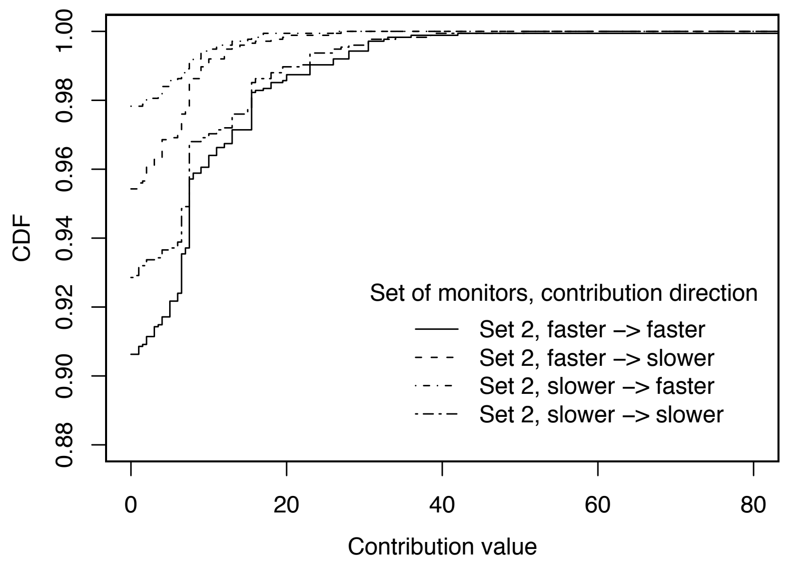
Accordingly, this index means that the total number of reduced j's probes thanks to the stop set entries originated by i.

Table 4.1 shows the average values of the contribution index in three iterations for respective monitor sets. Seeing the total values of the contribution index, we do not find the significant difference among them. Therefore, a mixture of slower monitors scarcely affected the efficiency of topology discovery. Paying attention to a probing pace in the case of Set 2, though the contribution from faster nodes $\left(C_{f\to f} + C_{f\to s}\right)$ is slightly larger than one from slower nodes $\left(C_{s\to f} + C_{s\to s}\right)$, we can see that the slower monitors still make a considerable contribution of increasing the efficiency of topology discovery. In this table, N_i denotes the number of probes that i actually performed. Comparing the $C_{i\to j}$ values with the N_i values, regardless of a probing pace, we can say that any monitor can make a contribution to the reduction of the number of probes.

For looking into more details of the contribution index, in Fig. 4.6(a), we present the cumulative distribution function (CDF) of the contribution values in respective sets of monitors. In the case of Set 2, the roles of monitors (faster or slower) are not distinguished. One can see that about 80% of the stop sets were not utilized by the monitors in all the cases, and these graphs show similar tendency. Furthermore, there are rapid rises around the contribution value of 7 in these graphs. This is why the average values of H_g in respective cases are around 7, and unmeasurable H_g values are replaced with the average values. From the aspect of the overall contribution value, we cannot see a significant difference within this degree of the variation of the probing pace. Focusing on Fig. 4.6(b), which depicts the CDF of the contribution values with the distinction of the monitor roles, we can see that one monitor tends to make more contributions to monitors at



(a) Total contribution



(b) Contribution in each direction

Figure 4.6: Distribution of the contribution values.

the same pace, compared to ones to monitors at the different pace. For a concrete reason for this phenomenon, we will need longer-term and larger-scale observation.

Summarizing the characteristics of DTS and the experimental results in this section, we can say that any monitor probing at a slower pace does not intercept the actions of other monitors, and moreover, it can enough contribute to efficient probing in the overall system. Monitors with any characteristics, such as low performance and low-speed links, can arbitrary join DTS and contribute to topology discovery — this is the flexibility of DTS.

4.4.2 Robustness

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 (Sec. 4.4.2) and monitor failure (Sec. 4.4.2).

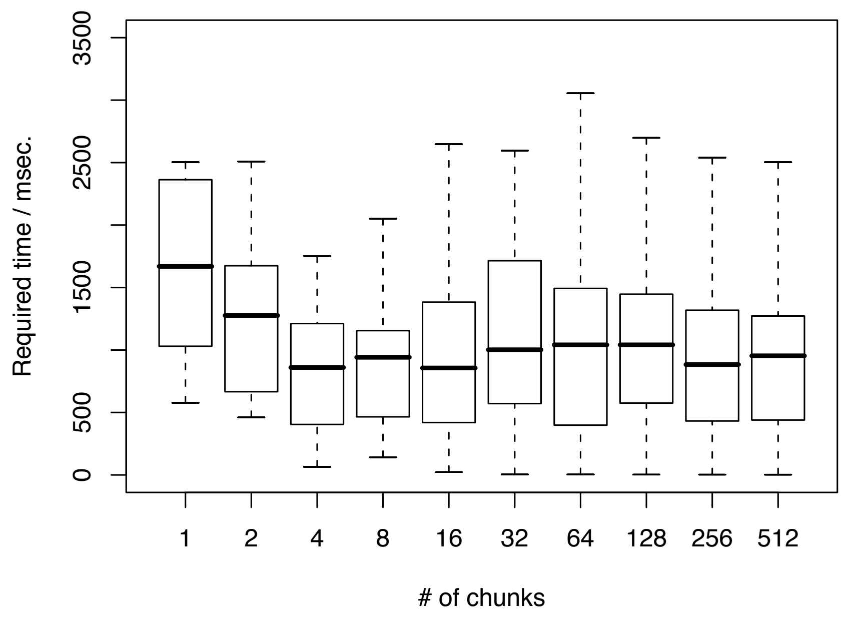
Impact of Chunk Size

As stated in Sec. 4.3.1, a merit concerning robustness is that DTS continues to work even if one or several monitors fail. This merit is brought by the separation of monitors and the shared database and the flow of procedures in the Doubletree algorithm. In a similar way presented in Sec. 4.4.1, if a monitor unexpectedly disappears from DTS, other monitors need to neither stop their work nor handle this matter as a special event.

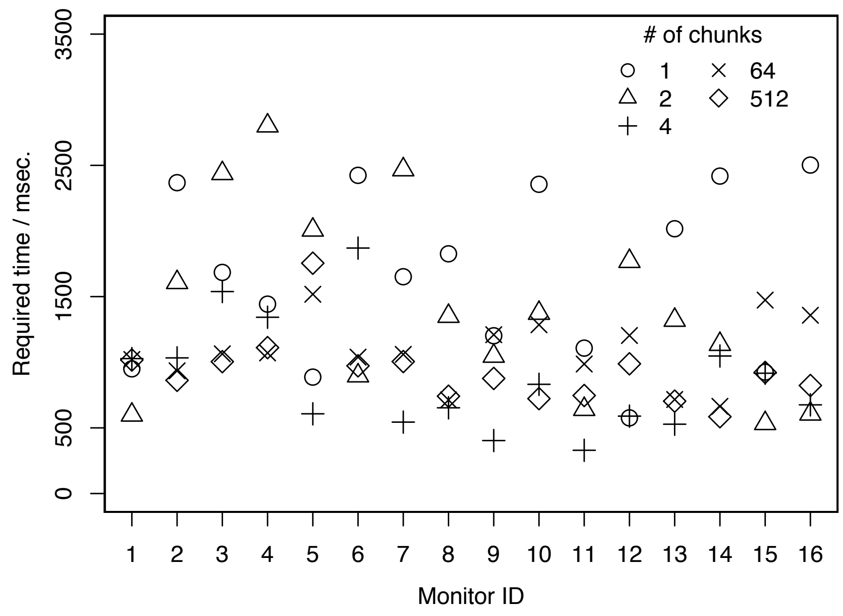
In fact, even though DTS can maintain its function, the failure of monitors causes the loss of data that are expected to be collected by the failed monitors. With the scheme of chunks, the impact of data loss depends on the size of chunks; the larger size of chunks will cause the loss of a larger part of data. 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, which is a part of the shared database. In respective cases, we made all monitors retrieve all chunks from the PT DHT and recorded the behavior of the monitors.

Fig. 4.7(a) 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 25th and 75th 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 25th percentile or 1.5 times IQR higher than the 75th percentile. One can see that the required time decreases as the



(a) Global time distribution



(b) Time distribution per monitor

Figure 4.7: Required time for retrieving one chunk from PT DHT.

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. The prior work on the performance evaluation of N-TAP in Chapter 3 also shows that it takes around one second for retrieving one database entry from the shared database with 16 core agents deployed on PlanetLab, and the required time is mainly dominated by the time for finding a responsible node in DHT. This fact matches the result of this experiment. We also note that, in the case that a monitor is a stub agent, a slight delay deriving from the communication with a core agent will be added to the required time, and the delay is negligible, compared to the required time for the overlay routing (see Chapter 3).

In addition, the required time for interacting with the DHT-based database is not homogeneous among monitors. Fig. 4.7(b) shows the average values of required time on each monitor for retrieving one chunk in the cases of $C = 1, 2, 4, 64, 512$. The average time does not necessarily show a consistent tendency according to the value of C. For example, the required time on the monitor #12 is the shortest where $C = 1$, though this value of C means the heaviest burden on the monitor among the values of C. One of the causes is that the required time depends on the status of each monitor, such as its CPU load and network connectivity, and the status changes dynamically due to the characteristics of the PlanetLab nodes as a shared testbed. Another cause that is particular to DTS is that, in DTS' DHTs that are constructed over N-TAP, an ID of a monitor is assigned randomly unless it is specified, therefore, the formation of the DHT network changed every iteration of the experiment. Consequently, there were both advantageous and disadvantageous monitors for retrieving chunks, which results in the large variance of the required time seen in Fig. 4.7(a). We have to consider that a smaller chunk does not necessarily bring the better responsiveness.

Thus, from these results, we should carefully choose the number of chunks, or the chunk size, on deploying DTS. If the number is too large, for example, C is 8 or more in Fig. 4.7, we cannot expect better responsiveness any more. Additionally, the larger number of chunks linearly increases the number of times of the interaction with the shared database since monitors commit and retrieve shared data in a unit of a chunk, and it results in the increment of the overall workload in DTS.

Impact of Monitor Failure

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.

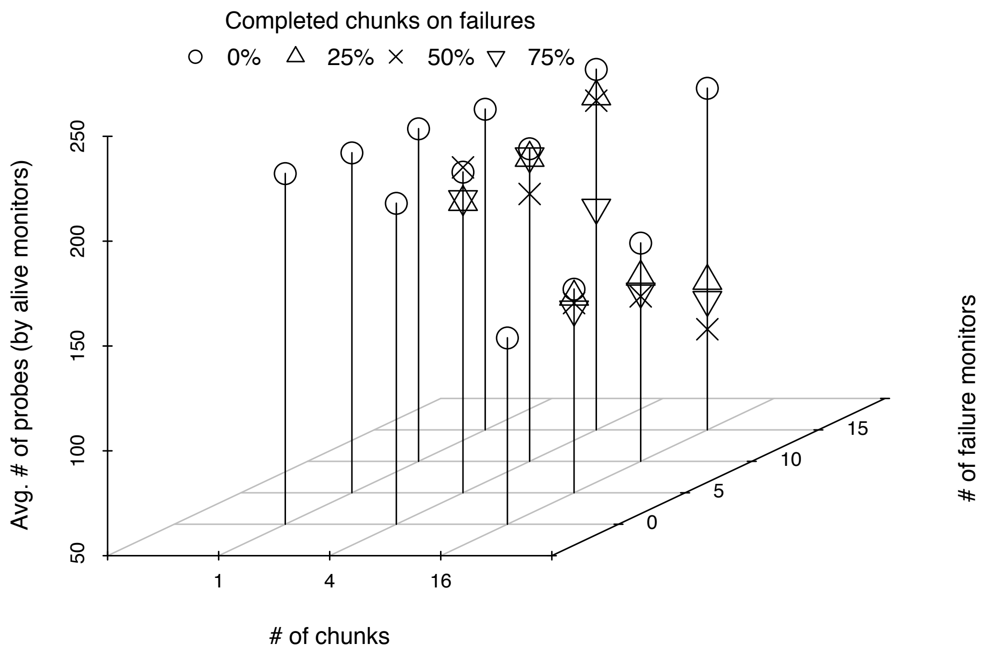


Figure 4.8: Impact of the failure of monitors and the chunk size on the number of probes.

For the experiment, we randomly chose 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 failure 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, finish topology discovery to the targets, we looked into the number of probes performed by the alive monitors.

Fig. 4.8 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 failure 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. Therefore, a monitor cannot expect the contribution to efficient probing from other monitors in this case.

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 that the failure 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 that 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.

From these results, we can say that smaller chunks will ensure less data loss and improve the efficiency of topology discovery even when monitor failures. However, as stated in Sec. 4.4.2, the smaller chunks will cause a heavier burden on the overall system. Considering this trade-off, it will be important to adjust the chunk size depending on respective DTS environments. For example, if one has high-speed links between monitors and the monitors have a sufficient performance, the chunk size can be smaller.

4.4.3 Scalability

DTS should scale against the increment of the number of monitors, and such increment mainly brings three considerations. One is the growth of the size of shared data, because the entries of the global stop set and collected topology data increase as the system becomes larger. The second one is the high frequency of the requests to the shared database. The last one is about the management of monitors.

On the first topic, focusing on the scalability of its shared database, we can simply consider that it is inherited from the one of N-TAP. N-TAP provides the Chord-based shared database [19] for measurement methodologies implemented on itself. DTS utilizes this feature for storing control data and collected data, i. e., use it for three DHTs presented in Sec. 4.3.1. Chapters 3 show that this shared database can take advantage of the nature of Chord. Indeed, the burden of holding shared data is evenly distributed among core agents. Further, the required time for finding an entry in the shared database, which is dominated by the time for the routing procedure in the Chord ring, increases in proportion to the logarithm of the number of core agents. Therefore, we can expect that the shared database scales just by adding a few core agents in the case that the size of shared data increases.

As the second topic, we have to consider that a larger number of monitors also brings more requests to the shared database. Since a DTS monitor always accesses to the shared database via a core agent with which the monitor connects, the workload will be concentrated on core agents. For scaling against such the workload, we will need some mechanisms that can distribute the workload among core agents, such as making a monitor randomly select a core agent with which the monitor is going to connect.

On the last topic, the management issue is easily solved thanks to DTS and the Doubletree algorithm. In the DTS architecture, there is no need for one entity to maintain and control monitors, and a monitor can arbitrarily join and leave the system as already described. Therefore, we can be released from such kind of tasks even if we have more number of monitors.

4.4.4 Modularity

As shown in Fig. 4.3 and Sec. 4.3.2, DTS is designed in a modular fashion, which divides a program into several independent components. Besides this fashion makes future extensions easier, it also enables the interconnection between DTS and other systems. Looking into the details, for example, a DHT Abstraction engine has a responsibility to connect DTS with a DHT service. Currently DTS utilizes the N-TAP's DHT-based database, as the result of making the DHT Abstraction engine to use the interfaces of N-TAP, such as the `storeInSharedDatabase()` method, which performs a procedure for storing a given object in the shared database. Even if we plan to use another DHT-based database, we just have to modify the parts of the interconnection in the DHT Abstraction engine, and basically, we do not need to touch other components such as Agent and Prober.

4.4.5 Revisit the Global Requirements

In this section, we evaluated the basic characteristics of DTS according to the global requirements stated in Sec. 4.3.1, that is, flexibility, robustness, scalability, and modularity. Finally, we summarize our investigations here.

On flexibility, we confirmed DTS ensures that a monitor can join and leave the system at will. Additionally, other merits are not only that a slower monitor does not disturb other monitors, but even a slower monitor can contribute to the overall efficiency of topology discovery in DTS. On robustness, we investigated the trade-offs between the chunk size and the workload on the system from the aspect of the interaction with DHTs and monitor failure. On scalability, we see that DTS can take advantage of the nature of the DHT-based database, and that DTS is superior in less burden of maintaining a number of monitors. On modularity, we confirmed that the design of DTS enables its easy extensions and flexible interaction with other systems.

4.5 Related Work

skitter [58], nowadays best known topology discovery tools, makes use of 24 monitors for tracing routes towards a set of 971,080 IPv4 addresses. scamper [73] behaves like skitter on a smaller scale and targets IPv6 networks. Other systems, such as RIPE NCC TTM [74] and NLANR AMP [75], consider a larger set of monitor, several hundreds, but avoid to trace outside their own network. A more recent tool, DIMES [49], is publicly released as a daemon. Rocketfuel [69] focuses on the topology of a given ISP and not on the whole Internet topology as skitter does, for instance. Finally, Scriptroute [68] is a system that allows an ordinary Internet user to perform network measurements from several distributed vantage points. It proposes remote measurement execution on PlanetLab nodes [41], through a daemon that implements ping, traceroute, hop-by-hop bandwidth measurement, and a number of other utilities. All of these systems operate under central control.

Unlike DTS, Rocketfuel and Scriptroute assume a centralized server to share stopping information (i. e., the list of previously observed IP addresses in RPT). Rocketfuel and Scriptroute do not consider how the information regarding where to stop probing can be efficiently encoded for exchange between monitors.

Measurement infrastructures are not limited to network topology discovery. NIMI [76] and Anemos [77], in the fashion of RIPE NCC TTM, work in full-mesh, i. e., measurements must be necessarily done between participants of the infrastructure. NIMI aims at facilitating the development of a large-scale measurement infrastructure for the Internet by using daemons as endpoints for a set of measurement tools. Anemos performs and analyzes active measurements on several network paths through a Web-based GUI. Unlike NIMI, Anemos also provides a way to collect measurements data into a MySQL database for later analysis, for instance. This later feature is somewhat similar to the DTM data plane. However, data storage in Anemos is centralized into a database and encounters thus the single point of failure risk, on the contrary to DTS where the data is stored in a distributed fashion.

Finally, the infrastructure might be a deployment facility. Examples of such a system are m-coop [78], pMeasure [55], and DipZoom [79]. These solutions are complementary to DTS in the sense that they can be used to distribute and manage DTS monitors.

4.6 Summary

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 chapter, that this sharing must be decentralized in order to be, among others, scalable and robust. In this chapter, we identified the needs of such a system.

This chapter proposed the first distributed topology measurement system, DTS, that is able to decentralize probing control, probing target and probing data information. In DTS, monitors cooperate through the use of three distributed hash tables, each one being dedicated to a particular probing information being shared. Through a deployment of DTS on the PlanetLab testbed, we demonstrated in this chapter that DTS meets distributed topology measurement systems: flexibility, robust, scalable, and modular.

Towards the actual deployment of a fully distributed topology measurement system, we still have several problems to be considered. These problems include a security matter regarding the authenticity of collected topology data, and a methodology of the actual operation. In fact, we cannot ignore such problems of the aspects untouched in this chapter and will need to tackle on them continuously.

Chapter 5

Towards the Deployment of an Application-Oriented Measurement Service

We presented several research contributions in the last chapters towards the ultimate goal, i.e., the actual deployment of an application-oriented measurement platform. In this chapter, looking back on the fundamental requirements for an application-oriented measurement platform and the ultimate goal of our study, we discuss the significance of our contribution, the future direction of application-oriented network measurement, and remaining issues.

5.1 Reviewing the Fundamental Requirements

In Section 2.3, we defined three fundamental requirements for application-oriented network measurement: responsiveness, accuracy, and coverage. Here we investigate whether the respective requirements can be met in an actual network environment through the review of our contributions.

5.1.1 Responsiveness

On the responsiveness of the application-oriented measurement service by N-TAP, its service time is mainly divided into these elements.

- Communication delay between an application and N-TAP
- Delay of querying on the DHT-based N-TAP network
- Time for performing a measurement procedure
- Computation time for calculating the target data
- Memory access time and storage access time

In the case of deploying N-TAP in the style of shared infrastructure described in Chapter 3, the most influential element is the delay of querying, considering the experiments in Chapter 3.

Basically, the DHT queries go across core nodes in multiple administrative domains, which result in high latency. In contrast, the communication delay between an application and N-TAP can be ignorable because its messages are exchanged within a local system or systems in the same domain. The measurement time and the computation time depend on the methodologies used for collecting the target network characteristics information, and N-TAP is not bound by any kind of measurement methodologies. By providing a general framework with which various measurement methodologies can be implemented, N-TAP will be able to choose an appropriate methodology among them for the collection. We can say that the memory and storage access time are generally ignorable compared to the other elements from the experiments.

Then we should consider how we lessen the impact of the query time. One is to construct the N-TAP network so as to cut down the query time. There exists some related work of speeding up the handle of queries [80], and autonomic overlay construction approaches for DHT introduced in Section 2.3. Applying these technologies to N-TAP, we can expect that its responsiveness is improved. Another approach is to reduce the number of queries, as we tried by creating caches on local systems. The impact of this approach depends on measurement methodologies and the bias of the measurement targets in which applications have interests. Additionally, the scale of N-TAP will have an impact, because if N-TAP is widely deployed and many applications utilize it, more duplication of required network characteristic information will rise among the applications. This will improve the reusability of once collected data, and the responsiveness itself. As another solution for reducing the number of overlay queries, increasing the number of entries in a finger table will be effective for the Chord-based overlay network, though it will consume more computation and network resources.

An entirely-different approach is to abandon a peer-to-peer network for the measurement network. Letting the N-TAP network have only one logical core agent, which is composed of synchronized multiple core agents, the service time will be improved owing to the shared data is accessible by one hop. However, the cost for synchronizing all shared data will be huge in the large-scale deployment, especially in the case of deploying the core agents in multiple domains as we assume for an AOMP. Constructing a tree-based network among core agents like DNS may work, but the measurement network does not have a chain of trust and responsibility as the DNS network. Therefore, maintaining a structure of such a tree-based network will produce an extra burden and may disturb the deployment. Considering these situations, we currently judge that a peer-to-peer network will suit for the large-scale measurement for autonomic applications. Additionally, since the queries can be parallel processed, the measurement request can be also issued in parallel. Taking this into account, a scheduling mechanism that increases the job parallelism will be required for the part of the command center.

Furthermore, a measurement methodology that performs continuous measurement will contribute to improve the responsiveness. Vivaldi [15] is a good example; it continuously measures RTT among a subset of target nodes and update the full-meshed RTT data. This allows the quick retrieve of target data because the target data is basically measured at the time when it is needed. The preference of continuous measurement methodologies may be a good tactic for the improvement of the responsiveness.

Totally, for the actual deployment of N-TAP, we will need to adopt additional techniques for the improvement of its responsiveness. And we already have some existing techniques, so trying on them can be future work. According to our experimental results, in a scale of $1000 - 10000$ core agents, the query time will be approximately a few seconds. We have to investigate how this time scale has an impact on the behavior of a real application through the actual utilization.

5.1.2 Accuracy

The accuracy of collected data on N-TAP mainly depends on the measurement methodologies used for the collection. Currently, we have the methodologies of ping, traceroute, Vivaldi and Doubletree [12]. Ping and traceroute are simple methodologies and their measurement results are accurate in terms of their approach. More specifically, ping gives the RTT of ICMP packets and traceroute gives forwarding paths of ICMP or UDP packets from a monitoring node to a specific destination. On Vivaldi and Doubletree, these two cooperative measurement methodologies increase the efficiency of their measurement with the estimation of the network characteristics information of some parts of measurement targets. Hence their estimated data may be less accurate compared to the actual measurement data. On another front, we can expect that both methodologies have considerable accuracy on their results according to the validation shown in the papers above. Therefore, the network characteristics information obtained from N-TAP has considerable accuracy. As the case of responsiveness, we should concentrate on creating a platform for various measurement methodologies so that more accurate measurement methodologies [81, 82, 83] can be implemented on it.

Focusing on the changes of the network characteristics information in time series, the collected data will lose its value as time goes by since autonomic applications basically require the network characteristics information reflecting the current state of the Internet. Therefore, the quick provision of measured data is one essential point. N-TAP basically gives back measurement results soon after they are acquired, so we can regard that N-TAP keeps this rule. On the other hand, each N-TAP agent has its own local caches of previously collected network characteristics information and utilizes them for the quick responsiveness even though they may lose their accuracy. At this time, the judge whether it is accurate enough for utilization for applications is transferred to the applications themselves by letting them specify the required freshness of given information. Since covering all the requirements on the freshness is unrealistic, we took this choice, i.e., the delegation to the applications. For the future, we will need a field test of the impact of the age of these local caches on applications' behavior.

5.1.3 Coverage

N-TAP distributes its agents among end systems and these systems can become monitoring points. In our deployment scenario, autonomic applications run N-TAP agents on its local system and utilize the local agents, all the nodes on which these applications are running can be involved in the measurement procedures of N-TAP. Considering that an autonomic application usually requires the network characteristic information related to the nodes on which the same application is running, this nature of N-TAP is an advantage. Moreover, if we need more monitoring points even though they are not application nodes, we just have to let the agent programs run on them. This simple procedure for deployment favors the expansion of measurement targets. For the wider coverage, we may have a choice of embedding a feature of an N-TAP agent into commodity operating systems. Of course, as the coverage become wider, the scalability of measurement procedures is required. For that, the implementation of sophisticated measurement methodologies cannot be negligible, as repeatedly stated.

5.2 Enhancements of End-System Capabilities

In this section, we summarize the enhancements of end-system capabilities brought by N-TAP. Our research contributions of the implementation of an actual AOMP system and a cooperative topology-discovery methodology on it are targeted for enhancing the end-system capabilities. Through reviewing the new capabilities and improved capabilities of end systems, we explore how future end systems can behave on the Internet.

One of the enhancements is that applications get a power of cooperative measurement methodologies. N-TAP enables one node to perform bidirectional measurement with a unidirectional measurement methodology. Owning to a feature of communicating with other agents, an N-TAP agent can ask them to perform the measurement, which is the simplest style of cooperative measurement. For instance, traceroute is a typical unidirectional measurement methodology which discovers only forwarding paths from an agent to destinations, and the agent cannot discover the paths from other agents to destinations by itself. However, by using the feature of communicating with other agents, an agent can obtain a forwarding path from other agent to a specific point with a request for traceroute. That is, a unidirectional measurement methodology can be a bidirectional one with the cooperation among the agents. This enhancement brings the extension of measurement coverage on one end system with ease; an application on an end system just has to issue a request for obtaining the target data. In the case of traceroute, the sources of forwarding paths are limited within the end systems on which N-TAP agents run, however, considering that an autonomic application is essentially interested in the IP topology information among the application nodes, this enhancement will be enough attractive for autonomic applications.

Another enhancement is the simplification of measurement procedures and the acquisition of a measurement platform. Existing applications have to have their own code or utilize program libraries for measurement. Aside from the problem that code-writing for measurement does not motivate programmers, the utilization of the libraries also leave some problems because we basically have separate libraries for respective measurement methodologies and programmers must have to learn their usages separately, which brings an extra burden. Meanwhile, N-TAP provides a unified platform for implementing measurement methodologies and a common interface to utilize these methodologies, which promotes the usage of the application-oriented measurement from applications. Eventually, we can say that, with N-TAP, applications get a simpler way for utilizing various network measurement methodologies as an improvement of end-system capabilities.

Thinking of these enhancements, the total enhancement can be expressed as a more capability of grasping the state of the Internet. Our approach is to enhance end-system the capabilities through the provision of a platform that eases the utilization and implementation of measurement procedures, and we have confirmed that applications running on end systems can easily obtain the network characteristics information of more targets with N-TAP. With further improvements of N-TAP, the actual deployment will be accelerated and we can expect to see the emergence of more capable network applications.

5.3 Deployment Scenarios

There will be several possible scenarios of deploying N-TAP on the Internet. Among them, we can find some trade-offs on some indices such as deployment cost and measurement accuracy. In

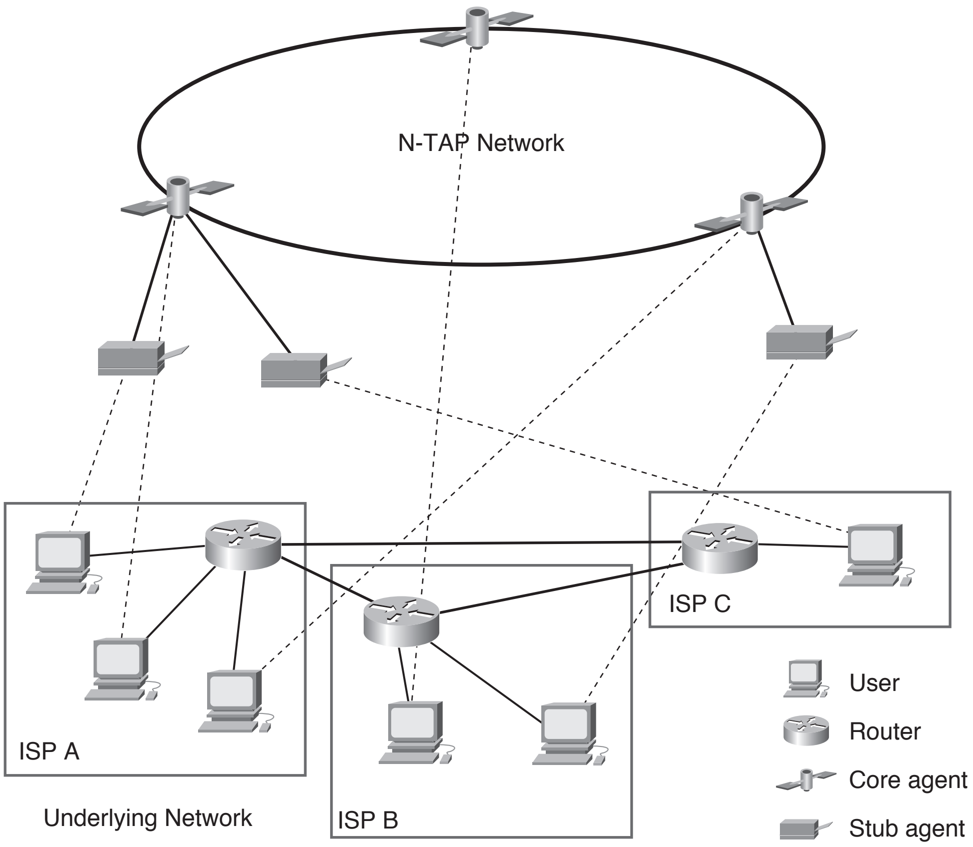


Figure 5.1: Deployment scenario: pure End-system-based service.

this section, we focus on these scenarios and discuss their merits and demerits.

The first possible scenario is to deploy N-TAP as a pure end-system-based service. Considering the architecture of N-TAP, this scenario makes all agents reside in the nodes on which applications are running regardless of agents' role, i.e., core or stub. Figure 5.1 shows this situation; all core and stub agents are distributed among end systems regardless of their locations on the underlying network. In this case, we have a merit of less deployment cost, because we do not need to prepare any fixed infrastructure such as nodes and networks. All we have to do is that the nodes on which autonomic applications are running let an N-TAP agent run on it, too. The applications basically utilize the agent running on a local node. At the same time, we have some difficulties for maintaining such kind of service. One problem is the stability of the overall service. We cannot preliminarily know which nodes are stable or unstable, therefore, if the system chose unstable nodes as ones on which core agents run, the stability of the overall system decreases. For the same reason, we will also have a performance problem such as worse responsiveness. In order to deploy an AOMP in this manner, an algorithm for choosing core agents and a mechanism of fault tolerance will be important factors.

The second scenario is to delegate core agents to ISPs and ordinary users run stub agents locally as depicted in Figure 5.2. In this case, ISPs prepare their own nodes for letting core agents run on them, and user nodes have their own stub agents connected to the core agents in the same or proximate ISP. We can assume that the ISP nodes are well maintained and have considerable resources such as computing power and broadband network. This leads the overall

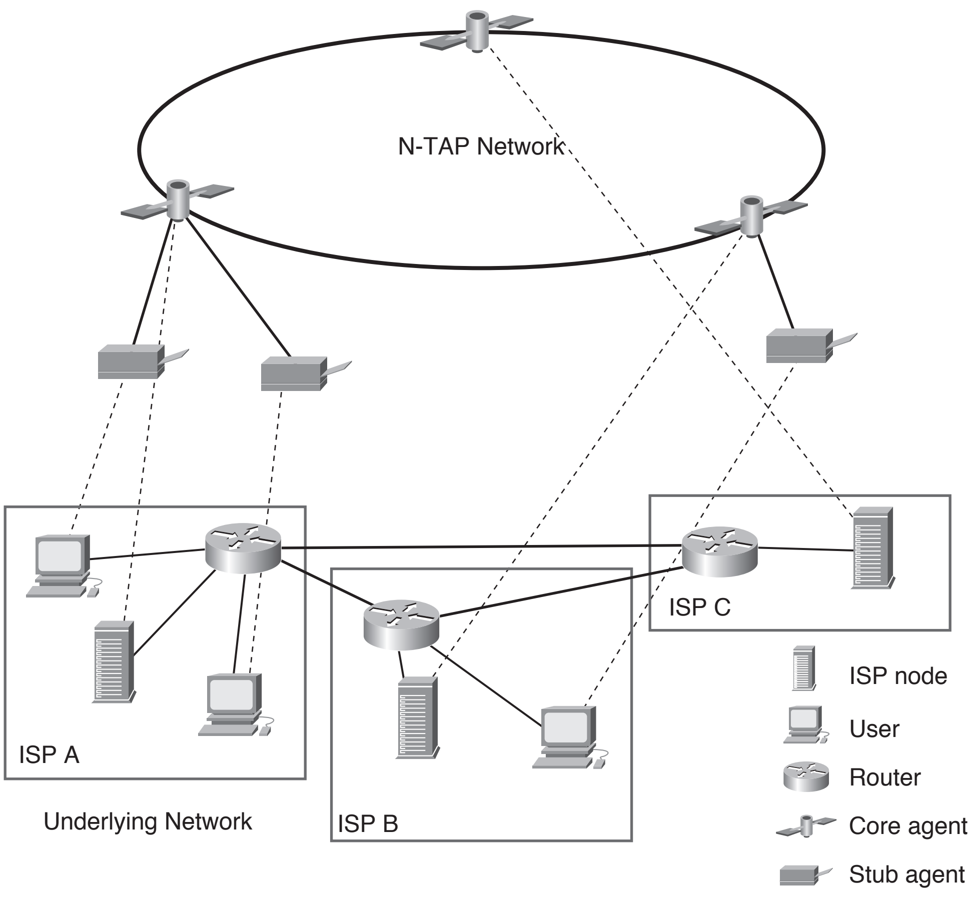


Figure 5.2: Deployment scenario: ISP-supported core agents.

system to be considerably stable, compared to the scenario of the pure end-system-based system. Additionally, since the stub agents are connected to proximate core agents, we can expect that the performance degradation caused by network latency will be reduced. However, the rise of deployment cost derived from the arrangement and management of ISP nodes is a demerit.

As a derivative scenario from the second one, we can also suppose that we remove measurement functions from the stub agents and just let them ask core agents for measurement. To achieve this scenario, we have to consider the granularity of network characteristics information. We can expect that some kinds of network characteristics information such as RTT between one node and a specific node are almost equal with the same ISP. In this case, only one node in one ISP has to perform measurement procedures and share the results among the nodes within the same ISP. Though we need further validation for such inference method, this scheme will be able to drastically reduce measurement overhead caused by an AOMP. Considering that an AOMP is deployed over the wire-area networks, this approach will be worth as an inference method for large-scale measurement.

5.4 Open Issues

For the actual deployment of AOMPs, we can find several remaining issues that should be solved before its full-scale deployment. At the last of our discussion, we inscribe them as future work.

5.4.1 Fault Tolerance

As an infrastructure system on the Internet, the fault tolerance of an AOMP is essential. In the current design, N-TAP is weak against the crash of a core agent. By the crash, N-TAP will lose some parts of collected measurement data and a crashed agent as a measurement resource. Additionally, the churn of the core network brings the service unavailability to a part of end systems. This defect comes from the difficulty of maintaining an overlay network based on the Chord algorithm. We already have several existing techniques for avoiding such situation. The replication of data entries on non-responsible nodes, more finger table entries on each node, and other improvements of DHT-based systems can also be applied to N-TAP. In future, we will need to explore a more stable methodology for constructing an overlay network.

5.4.2 Unimplementable Measurement Methodologies

We cannot leverage the structure of the measurement network for the measurement itself as a system introduced by Tagami et al. does [84]. Their system additionally measures RTT among overlay nodes in a Chord ring while a query traverses the nodes. Currently, N-TAP conceals the internal structure of the measurement network to the command center, i.e., implementers of measurement methodologies, however, if a measurement methodology that leverages such the structural information has good characteristics for application-oriented network measurement, it will be worth of considering the export of such information. On the other hand, such a modification will corrupt the division of roles in an AOMP and lose the generality of writing code on the command center, because such a measurement methodology only works on a specific measurement network and we cannot replace the measurement network without discarding the methodology. Reviewing the current designs of N-TAP and a general AOMP will be needed for that problem.

5.4.3 Incorrect Network Characteristics Information

In the current design of N-TAP, the agents are assumed to work correctly and not to perform malicious activities. Therefore, in the case that a broken agent malfunctions or a malicious agent appears and it provides incorrect network characteristics information for applications, an application will be misled to unexpected behaviors. The misbehavior of applications can be a big problem especially in the case of the large-scale applications like existing autonomic applications. One bad scenario will be an attack to systems with poor resources on the Internet. Falsifying measurement data so that a system with poor resources looks like having rich resources, applications that received the falsified data will try to aggressively utilize the resource of that system, which results in the overload on the system. In order to handle this kind of situation, we will have to prepare an authentication method among the agents and a mechanism of cross-checking of collected data by third-party agents.

5.4.4 Actual Utilization from a Real Autonomic Application

Though we have learned that our system will work correctly as designed and have considerable perform and features so far, the field test with a real autonomic application is still remaining. Several unexplored topics such as an impact of the responsiveness and accuracy on the behavior of the applications should be rapidly investigated before the actual deployment.

5.5 Future Direction

Recently we have seen more emphasis on the importance of grasping the Internet by applications. There really exists a trend of the infrastructures and services of collecting and providing network characteristics information [85, 86]. Even though these studies take different approaches for their goal, the goal is same as ours, i.e., the enhancement of end-system capabilities with the network characteristic information. Our contribution is not much more than one of the approaches, however, people are surely stepping forward towards the direction of the Internet with its clearer vista.

Additionally, when more autonomic applications appear and an AOMP is deployed wider, it will have more advantages. For example, more autonomic applications mean more overlaps of their related nodes and networks. This will increase the reusability of collected network characteristics information, which results in the better responsiveness for the provision of the information. However, while the scale of an AOMP is small, people may not find a great merit of an AOMP. Believing the appearance of more autonomic applications and that the better visibility of the Internet brings a crucial networking environment, we will have to prepare an infrastructure for an AOMP service and the enlightenment to ordinary users, application developers, and network administrators.

Even though the empowerment of end systems is like a two-edged blade, i.e., some people may be afraid that powerful end systems could destruct the core network on the Internet, however, we believe that such an evolution brings more profits rather than chaos with our self-control. Such the empowerment may allow end systems to detect currently hidden vulnerabilities on the Internet such as poor links. In history, radical changes on the Internet has brought temporal tragedies such as traffic overload by peer-to-peer file sharing applications, the shortage of IPv4 addresses by the rapid deployment of the Internet, and so on. Despite that, we have overcome the crises of the collapse of the Internet from the sides of management, operation, education and politics. For a capability of grasping the Internet on end systems, we should make a progress with care.

Network measurement has already been a casual procedure for applications, not a niche for researchers and network operators. Now it's time to unveil the internal of the Internet by Internet observation.

Chapter 6

Conclusions

In this dissertation, we first described the advantages and disadvantages of the current Internet based on its design principle, i.e., the end-to-end principle, and introduced emerging requirements for the Internet from autonomic applications such as a peer-to-peer network application. As a solution to the rise of such requirements, we defined a novel paradigm of network measurement called application-oriented network measurement, which collects and provides network characteristics information for these applications. The application-oriented network measurement is mainly required to have quick responsiveness, the accuracy of measurement data, and a wide-area coverage. Considering the wide and large-scale deployment, our approach is to provide the application-oriented network measurement as an independent network service. On that basis, we designed an application-oriented measurement platform (AOMP) by defining its fundamental features: service interface, command center, and measurement network.

According to the design, we implemented an AOMP built on a role-based peer-to-peer network called N-TAP, and evaluated its performance and scalability in actual network environments. The elemental component of N-TAP is an N-TAP agent, which has full features of an AOMP and cooperates with other agents for collecting network characteristics information. The N-TAP agents are divided into two roles: core and stub. Core agents construct and maintain the Chord-based measurement network, called the core network, for the management of measurement resources. Stub agents do not need to be involved in this operation, but both core and stub agents cooperate for measurement. As the result of the experiments in actual network environments, we obtained the following findings: (1) The N-TAP's core network has good scalability owing to the Chord algorithm. We can theoretically expect to have 1,000 core agents in the N-TAP system. (2) The query time on the core network is dominant in the entire service time of N-TAP. (3) The core agents must be stable and have much resources for the responsiveness and stability of the N-TAP service. The stub agents do not need to have the stability and rich resources, and they can even contribute in measurement and provide the measurement service for applications.

Moreover, we implemented one of the cooperative measurement methodologies called Doubletree, which discovers IP topology with avoiding redundant probes to the parts of networks that are already visited by other monitoring nodes. Our Double-tree based tracing system called DTS has several advantages. First, the utilization of the Doubletree algorithm decreases the number of probes and brings the quick grasp of the wide-area IP topology. This advantage is efficient especially in the case of the large-scale measurement required by autonomic applications. Secondly, the assignment of stub agents on all monitoring nodes, the failure of the monitoring nodes has little impact on DTS. In other words, a monitoring node can have a flexibility of arbitrary joining and leaving on this system. Thirdly, even slower monitoring nodes can contribute in topology discovery on that system. We confirmed that by the contribution index that indicates how many probes were reduced by the probe data collected by each node; the contribution index of slower nodes was just a bit less than that of faster nodes. This also proves the efficiency of the Doubletree algorithm in an actual network environment. We also explored a trade-off between the size of a chunk, which is a unit of managing the destinations list, and the data loss on the failure of monitor. And totally, we succeeded in an implementation of the cooperative measurement methodology on N-TAP with our design scheme.

Finally, reviewing our contributions, we discussed the merits and remaining problems of N-TAP. On the service responsiveness, the accuracy of collected data, and the coverage of measurement targets, we have considerably done for meeting respective requirements and suppose that we still require minor improvements of N-TAP for its actual deployment. Our goal, enhancing end-system capabilities of grasping the state of the Internet, is partially achieved and we could find some open issues for the complete accomplishment of our goal. Believing that clearer vista of the Internet counts for everyone, we continue to move forward step by step.

Bibliography

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, November 1984.

[2] J. Postel. Internet Protocol. RFC 791, September 1981.

[3] J. Postel. Transmission Control Protocol. RFC 793, September 1981.

[4] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the Internet: the end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology (TOIT)*, pages 70–109, August 2001.

[5] AKARI: Architecture Design Project for New Generation Network. http://akari-project.nict.go.jp/.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.

[7] Simon Patarin and Mesaac Makpangou. Pandora: An Efficient Platform for the Construction of Autonomic Applications. In *Self-star Properties in Complex Information Systems*, volume 3460 of *Lecture Notes in Computer Science (LNCS)*, pages 291–306. Springer, May 2005.

[8] J. Moy. OSPF Version 2. RFC 2328, April 1998.

[9] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable Application Layer Multicast. In *Proceedings of ACM SIGCOMM Conference 2002*, pages 205–217, August 2002.

[10] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Microsoft Research, 2003.

[11] Vinay Aggarwal, Anja Feldmann, and Christian Scheideler. Can ISPs and P2P Users Cooperate for Improved Performance? *ACM SIGCOMM Computer Communication Review (CCR)*, 37(3):31–40, July 2007.

[12] Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Efficient Algorithms for Large-Scale Topology Discovery. In *Proceedings of ACM SIGMETRICS Conference 2005*, pages 327–338, June 2005.

[13] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP). RFC 1157, May 1990.

[14] J. Postel. Internet Control Message Protocol. RFC 792, September 1981.

[15] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings of ACM SIGCOMM Conference 2004*, pages 15–26, August 2004.

[16] Brian Eriksson, Paul Barford, and Robert Nowak. Network Discovery from Passive Measurements. In *Proceedings of ACM SIGCOMM Conference 2008*, pages 291–302, August 2008.

[17] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM Conference 2001*, pages 161–172, August 2001.

[18] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-Aware Overlay Construction and Server Selection. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, pages 1190–1199, June 2002.

[19] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, February 2003.

[20] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.

[21] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In Rachid Guerraoui, editor, *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, volume 2218 of *Lecture Notes in Computer Science (LNCS)*, pages 329–350. Springer, November 2001.

[22] Zhichen Xu, Chunqiang Tang, and Zheng Zhang. Building Topology-Aware Overlays Using Global Soft-State. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)*, pages 500–508, May 2003.

[23] Mojtaba Hosseini, Dewan Tanvir Ahmed, Shervin Shirmohammadi, and Nicolas D. Georganas. A Survey of Application-Layer Multicast Protocols. *IEEE Communications Surveys and Tutorials*, 9(3):58–74, September 2007.

[24] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable Multicasting with on Overlay Network. In *Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI 2000)*, October 2000.

[25] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The Design of a Large-Scale Event Notification Infrastructure. In *Networked Group Communication [Proceedings of Third International COST264 Workshop (NGC'2001)]*, volume 2233 of *Lecture Notes in Computer Science (LNCS)*, pages 30–43, 2001.

[26] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A Case for End System Multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, October 2002.

[27] Sherlia Y. Shi and Jonathan S. Turner. Multicast Routing and Bandwidth Dimensioning in Overlay Networks. *IEEE Journal on Selected Areas in Communications*, 20(8):1444–1455, October 2002.

[28] Junghee Han, David Watson, and Farnam Jahanian. Topology Aware Overlay Networks. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, volume 4, pages 2554–2565, March 2005.

[29] Akamai Technologies. Akamai. http://www.akamai.com/.

[30] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP 2001)*, pages 131–145, October 2001.

[31] Akihiro Nakao, Larry Peterson, and Andy Bavier. A Routing Underlay for Overlay Networks. In *Proceedings of ACM SIGCOMM Conference 2003*, pages 11–18, August 2003.

[32] Srinivasan Seetharaman and Mostafa Ammar. Overlay-Friendly Native Network: A Contradiction in Terms? In *Proceedings of the Fourth Workshop on Hot Topics in Networks (HotNets-IV)*, November 2005.

[33] Lili Qiu, Yang Richard Yang, Yin Zhang, and Scott Shenker. On Selfish Routing in Internet-Like Environments. In *Proceedings of ACM SIGCOMM Conference 2003*, pages 151–162, August 2003.

[34] Yunhao Liu, Zhenyun Zhuang, Li Xiao, and Lionel M. Ni. A Distributed Approach to Solving Overlay Mismatching Problem. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS '04)*, pages 132–139, March 2004.

[35] Thomas Karagiannis, Pablo Rodriguez, and Konstantina Papagiannaki. Should Internet Service Providers Fear Peer-Assisted Content Distribution? In *Proceedings of Internet Measurement Conference 2005 (IMC 2005)*, pages 63–76, October 2005.

[36] Go Hasegawa, Masayoshi Kobayashi, Masayuki Murata, and Tutomu Murase. "Free-riding" Traffic Problem in Routing Overlay Networks. In *Proceedings of the 15th IEEE International Conference on Networks (ICON 2007)*, pages 118–123, November 2007.

[37] Cooperative Association for Internet Data Analysis. CAIDA : home. http://www.caida.org/.

[38] Kazaa. http://www.kazaa.com/.

[39] SQLite Home Page. http://www.sqlite.org/.

[40] Dave Winer. XML-RPC Specification. http://www.xmlrpc.com/spec, June 2003.

[41] The PlanetLab Consortium. PlanetLab — An open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org/.

[42] Toshiyuki Miyachi, Ken ichi Chinen, and Yoichi Shinoda. StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software. In *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools (VALUETOOLS 2006)*, October 2006.

[43] Nathaniel Leibowitz, Matei Ripeanu, and Adam Wierzbicki. Deconstructing the Kazaa Network. In *Proceedings of the Third IEEE Workshop on Internet Applications (WIAPP '03)*, pages 112–120, June 2003.

[44] T. S. Eugene Ng and Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, pages 170–179, June 2002.

[45] T. S. Eugene Ng and Hui Zhang. A Network Positioning System for the Internet. In *Proceedings of USENIX Annual Technical Conference 2004*, pages 141–154, June 2004.

[46] Marcelo Pias, Jon Crowcroft, Steve Wilbur, Tim Harris, and Saleem Bhatti. Lighthouses for Scalable Distributed Location. In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, volume 2735 of *Lecture Notes in Computer Science (LNCS)*, pages 278–291. Springer, October 2003.

[47] Manuel Costa, Miguel Castro, Antony Rowstron, and Peter Key. PIC: Practical Internet Coordinates for Distance Estimation. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS '04)*, pages 178–187, March 2004.

[48] Puneet Sharma, Zhichen Xu, Sujata Banerjee, and Sung-Ju Lee. Estimating Network Proximity and Latency. *ACM SIGCOMM Computer Communication Review (CCR)*, 36(3):39–50, July 2006.

[49] Yuval Shavitt and Eran Shir. DIMES: Let the Internet Measure Itself. *ACM SIGCOMM Computer Communication Review (CCR)*, 35(5):71–74, October 2005.

[50] The DIMES Team. The DIMES project. http://www.netdimes.org/.

[51] The NETI@home team. NETI@home. http://www.neti.gatech.edu/.

[52] Charles Robert Simpson, Jr. and George F. Riley. NETI@home: A Distributed Approach to Collecting End-to-End Network Performance Measurements. In *Proceedings of the 5th Anuual Passive and Active Measurement Workshop (PAM 2004)*, April 2004.

[53] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11):56–61, November 2002.

[54] University of California. SETI@home. http://setiathome.berkeley.edu/.

[55] Wenli Liu and Raouf Boutaba. pMeasure: A peer-to-peer measurement infrastructure for the internet. *Computer Communications*, 29(10):1665–1674, June 2006.

[56] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An Information Plane for Distributed Services. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)*, number 367-380, November 2006.

[57] Praveen Yalagandula, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Sung-Ju Lee. S^3: A Scalable Sensing Service for Monitoring Large Networked Systems. In *Proceedings of ACM SIGCOMM Workshop on Internet Network Management (INM'06)*, September 2006.

[58] Bradley Huffaker, Daniel Plummer, David Moore, and k claffy. Topology discovery by active probing. In *Proceedings of Symposium on Applications and the Internet (SAINT) 2002*, 2002.

[59] Van Jacobson. traceroute source code. ftp://ftp.ee.lbl.gov/traceroute.tar.gz.

[60] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proceedings of ACM SIGCOMM Conference 1999*, pages 251–262, August 1999.

[61] Jean-Jacques Pansiot and Dominique Grad. On Routes and Multicast Trees in the Internet. *ACM SIGCOMM Computer Communication Review (CCR)*, 28(1):41–50, January 1998.

[62] Anukool Lakhina, John W. Byers, Mark Crovella, and Peng Xie. Sampling Biases in IP Topology Measurements. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, pages 332–341, March 2003.

[63] Dimitris Achlioptas, Aaron Clauset, David Kempe, and Cristopher Moore. On the Bias of Traceroute Sampling: or, Power-law Degree Distributions in Regular Graphs. In *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC 2005)*, pages 694–703, May 2005.

[64] P. Erdos and A. Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.

[65] Bill Cheswick and Steve Branigan. Mapping and Visualizing the Internet. In *Proceedings of USENIX Annual Technical Conference 2000*, 2000.

[66] Neil Spring, David Wetherall, and Thomas Anderson. Reverse Engineering the Internet. *ACM SIGCOMM Computer Communication Review (CCR)*, 34(1):3–8, January 2004.

[67] kc claffy, Mark Crovella, Timur Friedman, Colleen Shannon, and Neil Spring. Community-Oriented Network Measurement Infrastructure (CONMI) Workshop Report. *ACM SIGCOMM Computer Communication Review (CCR)*, 36(2):41–48, April 2006.

[68] Neil Spring, David Wetherall, and Tom Anderson. Scriptroute: A Public Internet Measurement Facility. In *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS) 2003*, 2003.

[69] Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring ISP Topologies with Rocketfuel. *IEEE/ACM Transactions on Networking (TON)*, 12(1):2–16, February 2004.

[70] Marcel Karnstedt, Kai-Uwe Sattler, Manfred Hauswirth, and Roman Schmidt. UniStore: Querying a DHT-based Universal Storage. In *Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE) 2007*, pages 1503–1504, April 2007.

[71] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient Replica Maintenance for Distributed Storage Systems. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI '06)*, pages 45–58, May 2006.

[72] Yatin Chawathe, Sriram Ramabhadran, Sylvia Ratnasamy, Anthony LaMarca, Scott Shenker, and Joseph Hellerstein. A Case Study in Building Layered DHT Applications. In *Proceedings of ACM SIGCOMM Conference 2005*, pages 97–108, August 2005.

[73] IPv6 Scamper. http://www.wand.net.nz/ mluckie/ipv6-scamper/.

[74] Fotis Georgatos, Florian Gruber, Daniel Karrenberg, Mark Santcroos, Henk Uijterwaal, and Rene Wilhelm. Providing Active Measurements as a Regular Service for ISPs. In *Proceedings of Workshop on Passive and Acrive Measurements (PAM) 2001*, April 2001.

[75] A. McGregor, H. W. Braun, and J. Brown. The NLANR Network Analysis Infrastructure. *IEEE Communications Magazine*, 38(5), 2000.

[76] Andrew Adams, Jamshid Mahdavi, Matthew Mathis, and Vern Paxson. Creating a Scalable Architecture for Internet Measurement. In *Proceedings of INET 1998*, July 1998.

[77] Antonios Danalis and Constantinos Dovrolis. ANEMOS: An Autonomous NEtwork MOnitoring System. In *Proceedings of Passive and Active Measurement Workshop (PAM) 2003*, 2003.

[78] Sridhar Srinivasan and Ellen Zegura. Network Measurement as a Cooperative Enterprise. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, volume 2429 of *Lecture Notes in Computer Science (LNCS)*, pages 166–177, March 2002.

[79] Zhihua Wen, Sipat Triukose, and Michael Rabinovich. Facilitating Focused Internet Measurements. In *Proceedings of ACM SIGMETRICS Conference 2007*, pages 49–60, June 2007.

[80] Xin Li, Fang Bian, Hui Zhang, Christophe Diot, Ramesh Govindan, Wei Hong, and Gianluca Iannaccone. MIND: A Distributed Multi-Dimensional Indexing System for Network Diagnosis. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, pages 1–12, April 2006.

[81] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clemence Magnien, and Renata Teixeira. Avoiding Traceroute Anomalies with Paris Traceroute. In *Proceedings of Internet Measurement Conference (IMC) 2006*, October 2006.

[82] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring Load-balanced Paths in the Internet. In *Proceedings of Internet Measurement Conference (IMC) 2007*, pages 149–160, October 2007.

[83] Matthew Luckie, Young Hyun, and Brad Huffaker. Traceroute probe method and forward IP path inference. In *Proceedings of Internet Measurement Conference (IMC) 2008*, October 2008.

[84] Atsushi Tagami, Teruyuki Hasegawa, Shigehiro Ano, and Toru Hasegawa. Evaluation of P2P-Based Internet Measurement System on Loss Tolerance to Measurement Results. In *Proceedings of SAINT2007 Workshop on Internet Measurement Technology and its Applications to Building Next Generation Internet*, January 2007.

[85] Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovi, and Antonio Nucci. Unconstrained Endpoint Profiling (Googling the Internet). In *Proceedings of ACM SIGCOMM Conference 2008*, pages 279–290, August 2008.

[86] Haiyong Xie, Y. Richard Yang, Arvind Krishnamurthy, Yanbin Liu, and Abraham Silberschatz. P4P: Provider Portal for Applications. In *Proceedings of ACM SIGCOMM Conference 2008*, pages 351–362, August 2008.

PUBLICATION LIST

This section contains the information on the author's publications and related research projects as of March 2009. The items marked with asterisks (∗) have particular relevance to this doctoral dissertation.

Papers in Refereed Journals

- ∗ 益井賢次，門林雄基．階層型 P2P ネットワークを用いたインターネット計測基盤の性能評価と実展開シナリオの考察．情報処理学会論文誌， Vol. 50，No. 2，pp. 709–720，2009 年 2 月.

Papers in International Conferences and Workshops

- ∗ Kenji Masui and Benoit Donnet, "DTS: a Decentralized Tracing System." In Proceedings of the 1st International Workshop on Traffic Monitoring and Analysis (TMA'09), May 2009 (to appear).

- ∗ Kenji Masui and Youki Kadobayashi, "Observing the Dynamics of the Internet on Our Laptops with an Application-Oriented Measurement Platform." In Proceedings of ACM SIGCOMM Conference 2008, Demo Session, August 2008.

- ∗ Kenji Masui and Youki Kadobayashi, "A Role-Based Peer-to-Peer Approach to Application-Oriented Measurement Platforms." In Sustainable Internet [Proceedings of the Third Asian Internet Engineering Conference (AINTEC 2007)], Vol. 4866 of Lecture Notes in Computer Science (LNCS), pp. 184–198. Springer, November 2007.

- Yuzo Taenaka, Kenji Masui, Kimihiro Suwa, Khamphao Sisaat, Shigeru Kashihara, Takeshi Okuda, Youki Kadobayashi, and Suguru Yamaguchi, "Investigation of the Basic Characteristics of Long Distance Wireless LANs." In Proceedings of the Third International Workshop on Wireless Network Measurement (WiNMee 2007), April 2007.

- ∗ Kenji Masui and Youki Kadobayashi, "Bridging the Gap between PAMs and Overlay Networks: a Framework-Oriented Approach." In Passive and Active Network Measurement [Proceedings of the Eighth Passive and Active Measurement Conference (PAM 2007)], Vol. 4427 of Lecture Notes in Computer Science (LNCS), pp. 265–268. Springer, April 2007.

- ∗ Kenji Masui and Youki Kadobayashi, "N-TAP: A Platform of Large-Scale Distributed Measurement for Overlay Network Applications." In Proceedings of the Second International Workshop on Dependable and Sustainable Peer-to-Peer Systems (DAS-P2P 2007), January 2007.

- Kenji Masui, Shinya Nakamura, Masashi Eto, and Youki Kadobayashi. "Policy-Based Email System for Controlling Secondary Use of Information." In Proceedings of the First International Workshop on Security (IWSEC2006), October 2006.
- Yukio Okada, Kenji Masui, and Youki Kadobayashi, "Proposal of Social Internetworking." In Web and Communication Technologies and Internet-Related Social Issues [Proceedings of the 3rd International Human.Society@Internet Conference (HSI 2005)], Vol. 3597 of Lecture Notes in Computer Science (LNCS), pp. 114–124. Springer, July 2005.

Other Published Papers and Technical Reports

- Hieu Hanh Le, Masayoshi Shimamura, Kenji Masui, and Katsuyoshi Iida, "A Study on XCP Routers' Misbehaviors on Congestion Control." In Technical Report of IEICE, Vol. 108, No. 460, IA2008-75, pp. 49–54, March 2009.
- 三宅光太郎，益井賢次，飯田勝吉．プロキシ型モバイル通信におけるノードの位置情報漏洩の分析と許容遅延を考慮した対策の提案．電子情報通信学会技術研究報告，Vol. 108，No. 457，NS2008-229，pp. 483–488，2009 年 3 月．
- 田中彰，嶋村昌義，益井賢次，飯田勝吉．有害コンテンツ抑制を目的としたピアとコンテンツ双方にレピュテーションを用いる P2P コンテンツ流通手法の提案．電子情報通信学会技術研究報告，Vol. 108，No. 457，NS2008-168，pp. 141–146，2009 年 3 月．
- 大溝拓也，益井賢次，飯田勝吉．重複経路の削減を目的とする AS 間オーバレイ経路制御のためのクロスレイヤアーキテクチャの一検討．電子情報通信学会技術研究報告，Vol. 108，No. 258，NS2008-70，pp. 13–18，2008 年 10 月．
- ∗ 益井賢次，門林雄基．エンドノードからの IP トポロジ探索におけるノード検出回数に基づくトポロジ特性の分析．電子情報通信学会 技術研究報告，Vol. 108，No. 120，IA2008-14，pp. 7–12，2008 年 7 月．
- 妙中雄三，益井賢次，諏訪公洋，Khamphao Sisaat，樫原茂，奥田剛，門林雄基，山口英．長距離無線 LAN の特性を考慮した性能指標の検討．電子情報通信学会技術研究報告，Vol. 106，No. 578，IN2006-197，pp. 101–106，2007 年 3 月．
- ∗ 益井賢次，宮本大輔，門林雄基．ネットワーク特性共有のための分散型基盤の提案と設計．第 7 回インターネットテクノロジーワークショップ (WIT2005)，2005 年 11 月．
- 益井賢次，岡田行央，新井イスマイル，市川本浩，中村豊．ネットワークの一時利用を実現するコンポーネント独立で可搬性の高い利用者管理システムの設計と実装．第 9 回分散システム/インターネット運用技術シンポジウム，2004 年 12 月．

Research Grants

- ∗ 日本学術振興会，日本学術振興会特別研究員 (DC2)．アプリケーションの多様な要求を考慮したネットワーク特性収集方式の研究，2008 年 4 月．
- 情報処理推進機構，未踏ソフトウェア創造事業 共同開発者．オーバレイネットワークを用いた MMOG インフラストラクチャの開発，代表者 飯村卓司，2004 年度第 2 回．