Application of Overlay Techniques to Network Monitoring

by

Zhan Xiaoying

A dissertation submitted in fulfillment of the requirements for the degree of Doctor of Philosophy at the University of Glasgow

May 2008

Abstract

Measurement and monitoring are important for correct and efficient operation of a network, since these activities provide reliable information and accurate analysis for characterizing and troubleshooting a network's performance. The focus of network measurement is to measure the volume and types of traffic on a particular network and to record the raw measurement results. The focus of network monitoring is to initiate measurement tasks, collect raw measurement results, and report aggregated outcomes.

Network systems are continuously evolving: besides incremental change to accommodate new devices, more drastic changes occur to accommodate new applications, such as overlay-based content delivery networks. As a consequence, a network can experience significant increases in size and significant levels of long-range, coordinated, distributed activity; furthermore, heterogeneous network technologies, services and applications coexist and interact. Reliance upon traditional, point-to-point, *ad hoc* measurements to manage such networks is becoming increasingly tenuous. In particular, correlated, simultaneous 1-way measurements are needed, as is the ability to access measurement information stored throughout the network of interest.

To address these new challenges, this dissertation proposes OverMon, a new paradigm for edge-to-edge network monitoring systems through the application of overlay techniques. Of particular interest, the problem of significant network overheads caused by normal overlay network techniques has been addressed by constructing overlay networks with topology awareness - the network topology information is derived from interior gateway protocol (IGP) traffic, i.e. OSPF traffic, thus eliminating *all* overlay maintenance network overhead.

Through a prototype that uses overlays to initiate measurement tasks and to retrieve measurement results, systematic evaluation has been conducted to demonstrate the feasibility and functionality of OverMon. The measurement results show that OverMon achieves good performance in scalability, flexibility and extensibility, which are important in addressing the new challenges arising from network system evolution. This work, therefore, contributes an innovative approach of applying overly techniques to solve realistic network monitoring problems, and provides valuable first hand experience in building and evaluating such a distributed system.

Acknowledgement

First and foremost, I would like to thank my direct supervisor Professor Joe Sventek, for everything he has done for me: from daily professional guidance, to extended financial support, and the multiple readings of my dissertation in a very tense schedule. This dissertation would not have been possible without his persistent encouragement and constructive criticism. Especially, during the downtime I experienced in my last year of study, his preciseness in pursuing quality research, determination to overcome difficulties, and hardworking attitude set a vivid example for me to carry on and finish this PhD. His support and guidance throughout my PhD studies have had a lasting, positive impact on me, which will help me to face other difficulties and challenges in the future.

I also want to express my special gratitude to my second supervisor Dr Peter Dickman. I thank him for being the one giving me the urge to start writing, helping me to accumulate research skills in writing and presenting, and his supportive encouragement and mentoring when I was in dark moments. I appreciate indeed the constructive criticism and suggestions he gave after reviewing my dissertation, although by that time he had already left the department.

I'm grateful to my colleagues for their invaluable support in the department. Particularly, I thank Dr Colin Perkins for inspirational discussions of my work; Naveed Khan for his repeated installations of Xen software; Dr Jonathan Paisley for his initial help on using Linux; Dr Liangxiu Han for sharing her experience of using measurement tools; Dr Olufemi Komolafe for being the senior looking after the juniors (and particularly organizing group members around the entertaining lunch tables); Oliver Sharma for sharing his experience of capturing OSPF packets; Ross McIlroy for sharing his experience of using Xen software; Stephen Strowes for insightful feedback after reading my dissertation; Steven Heeps, Alexandros Koliousis, and Mohammed Aminu for light and witty chats.

I would like to thank my Chinese friends in Glasgow and in UK, Di Cai, Yan You, Yunan Zheng, Zhichao Wu, Ben He, Reede Ren, Chiew Thiam Kian, Chunxia Li, Min&Xiaodong Li, Martin&Suk-Ching Song, and others I could not possibly include them all in the lines. I thank these friends for their long-lasting friendship and encouragement - with them around, besides hard working, the PhD life is also blended in with pleasant memories on laughs, food, badminton, and travelling.

Studying abroad is a special experience in my life; my family makes this all more meaningful. I thank my dearest mum for bearing the hardship of bringing up three children

alone after my dad passed away, and her bravery in fighting with diseases over the years. I am in debt to my brother and sister for always being there providing me endless supports, and looking after my mum. I cherish my extended family and thank all my in-laws for their love and understanding. Lastly, I thank my husband Feng, who shares the most of my frustrations and struggles throughout my PhD study. I thank him for his unconditional love, courage, patience, and tolerance; and providing me a warm home full of happiness. Without these, my life would have been very different.

The work presented in this dissertation was partially funded by the UK Engineering and Physical Sciences Research Council (EPSRC) as part of the Performance Measurement and Management for Two-Level Optimization of Networks and Peer-to-Peer Applications (P2POpt) (Project No.GR/S68989/01).

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, following a brief introduction to the background of the dissertation, the motivation for this work is firstly discussed in Section 1.1. Then, the thesis statement and contributions are summarized in Section 1.2 and 1.3, respectively. The roadmap for the dissertation can be found in Section 1.4.

1.1 Motivation

To operate a network properly and efficiently, network management is very important. Generally speaking, network management is "a service that employs a variety of tools, applications, and devices to assist human network managers in monitoring and maintaining networks" [CiscoMng]. The International Standard Organization (ISO) has produced a network management model for understanding the major functions of network management systems. This model consists of five conceptual management areas, namely performance, configuration, accounting, fault, and security. This dissertation is focussed on performance management.

By "performance management" the core activities include planning, allocating and optimizing network resources based on the results obtained from network measurement and monitoring. In this context, network measurement and monitoring are fundamental components of network management, in the sense that they provide reliable information and accurate analysis in characterizing and troubleshooting a network's performance; this information has a significant impact on the tactics that are chosen to achieve the management strategy set up for the network. This dissertation focussed on network monitoring, since the testing environment that network monitoring provides is required to initiate required active network measurements and to retrieve and analyze the results of those measurements.

Current network systems are continuously evolving: besides incremental change to accommodate new devices, more drastic evolution is required to accommodate new applications that demand different qualities of service. Networks that can cope with these

new requirements are typically termed ‘next generation networks (NGNs)’. The evolution towards NGNs presents new challenges for the network monitoring community. Firstly, compared to the relatively static networks of the past with carefully planned topologies, current and future networks are expected to be dynamically constructed with a logical overlay layer in real time. As a consequence, the network size can easily increase by large amounts with a high level of distribution. Furthermore, heterogeneous network technologies, services and applications coexist and interact; the variety of these factors, along with the possibility of accessing them from virtually *any* location, make it extremely complicated to foresee the type, volume and distribution of network traffic.

In the face of dynamic topologies, rapidly increasing scale and significant complexity in future networks, conventional network monitoring approaches have difficulty in coping with these challenges due to the lack of *flexibility*, *scalability* and *extensibility*. For example, as the de facto standard, the Simple Network Management Protocol (SNMP) [SNMP98] is used by network management systems to monitor network-attached devices. However, SNMP is normally deployed at a fixed number of sites, and the monitoring functionalities are mostly static and predefined; it is easy to generate high volumes of measurement data and significant measurement overheads, which leads to server side bottlenecks and single points of failure. Although effort has been made to make a monitoring system decentralized and programmable - e.g. to employ a hierarchical infrastructure based on distributed objects [CORBAWeb, RMIWeb], the installation of these distributed objects often requires a significant amount of human administration; and for a large-scale, ever-changing network, the intelligence required to *locate* measurement sites and support the underlying communication infrastructure are still far from being mature [Goldszmidt95, Liotta02].

In this dissertation, it is envisioned that the new challenges brought by the drastic evolution of network technologies on the one hand aggravate the complexity and difficulty in managing today and future’s networks, on the other hand they provide potential advantages. Overlay networking is such a two-edged technology. By using overlay networks, end hosts can sidestep the central server in a client-server model and connect to each other directly; new network services and functionalities can be created quickly and easily; and these new changes do not require universal support or adoption from the underlying network layer [BitTorrentWeb, NapsterWeb, eDonkeyWeb, Chu00, and Pendarakis01].

However, overlay networks are normally built without knowledge of the underlying physical network, thus cannot have any direct control over how packets are routed in the underlying network between two overlay nodes. To improve the routing performance, the

most common way is to send probe packets to obtain underlying network information; obviously, this leads to inefficient usage of network resources, due to the large number of such control messages that are generated to maintain the overlay.

This dissertation is then motivated to solve this problem: 1) to cope with the new networking challenges, an efficient network monitoring system is required to be flexible to address dynamically changing topologies, scalable to address increasing network size, and extensible to address high levels of complexity and heterogeneity; 2) if the overhead that are normally introduced by overlay networks can be controlled, the advantages of overlay techniques can be leveraged to construct such a monitoring system.

1.2 Thesis Statement

Conventional network monitoring approaches have difficulty in addressing dynamic topologies, rapidly increasing scale and significant complexity in networks. There is a need for flexible, scalable, and extensible monitoring systems to address these issues. I assert that bandwidth-efficient overlay-network technologies can be designed, implemented and exploited to construct such a system and efficiently provide the required flexibility, scalability and extensibility.

1.3 Contribution

The primary technical contributions of this work are firstly architectural and evaluative, then algorithmic. These contributions are summarized as below:

- Identify a practical subset of network monitoring functionality that requires improved flexibility, scalability and extensibility;
- Critically analyse application of current overlay network technologies to network monitoring systems;
- Architectural and algorithmic design of bandwidth-efficient overlay techniques for use in network monitoring:
    - By snooping on interior gateway protocol (IGP) traffic among the routers in a network, e.g. OSPF traffic, one can reconstruct the topology of a network, which is important in maintaining and optimizing overlay structures, and normally, can only be obtained through periodic ping

message traffic;

- Exploit such IGP information to construct application-level multicast trees among edge routers of the network to facilitate measurement control traffic;
- Exploit such IGP information to construct application-level ring structures among edge routers of the network to facilitate access to accumulated measurement data.

- Experimental system design and performance study of these bandwidth-efficient overlay techniques applied to the subset identified above to demonstrate the efficacy of the approach. Of particular interest is the design of a Xen-based emulation framework, and a customised simulation solution for the experimentation.
- Enumerate a number of areas of future work based upon the contributions of this dissertation.

1.4 Dissertation Roadmap

The rest of this dissertation is organized as follows.

Chapter 2 covers a broad discussion of background information. It starts with a revisiting of the architectural and operational concepts of the Internet. Next, it introduces network measurement and monitoring for their importance, roles, and operational focus in network management, with SNMP as an example showing how network monitoring facilitates network management activities. Then, the discussion of overlay networks follows: firstly with fundamental concepts, then a case study of Peer-to-Peer (P2P) networks - the most representative paradigms of overlay networks.

Chapter 3 presents a critical analysis of the problem that is to be solved in this dissertation. It firstly discusses why an overlay based network monitoring system is needed; then a new network monitoring system, OverMon, is defined from different perspectives; follows with the discussion of relevant overlay technologies with respect to their high level introductions, taxonomies and their applicability to OverMon, and the related research work with respect to their relevance and difference to OverMon.

Based on the analysis of Chapter 3, in Chapter 4, the OverMon design is presented in detail: firstly with the high level design principles and the system infrastructure; then the major components, namely topology tracker, control overlay, and data overlay. With the topology

information provided by the topology tracker, the control overlay and data overlay can be constructed with topology-awareness, which not only saves the overhead caused by normal overlay maintenance, but also improves the overlay's performance by minimizing its impact on the underlying network.

Then, in Chapter 5, as the proof of the concept, the implementation details of OverMon are introduced, from a software engineering perspective, particularly focused on the design of internal and external interfaces.

The systematic evaluation of OverMon is presented in Chapter 6. It firstly discusses the general methodologies that are widely accepted to evaluate a large distributed system, and describes the proper strategies to evaluate OverMon. Next, a Xen-based emulation toolkit is introduced, by which the emulation networks used to evaluate OverMon can be set up or torn down easily and efficiently. Then detailed evaluation of the control overlay and the data overlay of OverMon are presented respectively, with respect to the measurement metrics, the experimental methodologies and the evaluation results.

Finally, in Chapter 7, conclusions are presented with a summary of the dissertation contributions, future work, and final remarks.

Chapter 2

Background

This chapter provides background information required for this dissertation. It starts with an overview of the current Internet's architecture and operation. More detailed knowledge regarding current approaches to network measurement and monitoring follows, with SNMP as an example. Next, overlay networks are introduced, with fundamental concepts first, then a case study of Peer-to-Peer networks (P2P).

2.1 Internet Architecture and Operation

This section provides an introduction to the organizational principles of the Internet, which are helpful in understanding the role of network monitoring systems, and how they are deployed in operational networks.

2.1.1 Packet Switched Networking

Conceptually, the Internet is a collection of packet-switched networks. In a packet-switched network, data to be moved from one system to another is first broken up into packets. Each packet carries enough information for an intermediate device to transport the packet towards its destination endsystem. *Endsystems* are packet sending and receiving devices such as a PC or a server. The intermediate devices are *routers* connected by *links*. Links physically transport a packet from place to place. Routers receive packets from incoming links and place them onto outgoing links; as a result, packets are directed and forwarded towards their destination.

The communicating of data based on packet-switching is governed by communication standards, i.e. protocols. For various subtasks, the protocols used in the Internet are usually

considered to be organized into four layers. Each layer has a particular responsibility and provides a service to the layer above it. The four layers are shown in Figure 2-1. Starting at the lowest layer, an overview of each layer follows:

- **Link** The link layer is normally associated with a particular communication medium. Link layer protocols manage how packets are physically transport along the medium from one location to another. Ethernet is an example of a link layer protocol.
- **Network** The network layer is largely associated with routers. Network layer protocols manage how packets are forwarded from incoming links to outgoing links towards their destination. The Internet Protocol (IP) is the only network layer protocol in the Internet, and is the most important protocol in the Internet. the IP protocol has two versions: IPv4 and IPv6; the former is predominant. IPv4 defines the format of router and endsystem addresses, i.e. *IP addresses*, which are expressed as four decimal numbers, each representing 8 bits of the address.
- **Transport** The network layer causes packets to be shipped from the source endsystem to the destination endsystem. Transport protocols connect processes on the two endsystems, and also determine the quality of service available to the applications. TCP and UDP are two major transport layer protocols.
- **Application** The application layer is concerned with implementing particular applications, such as the World Wide Web (WWW), the Domain Name System (DNS), e-mail etc. Typical application protocols include HTTP for WWW, DNS for DNS, and SMTP for e-mail.

To perform its functionality, each protocol needs to transmit control data. The control data takes the form of *headers*. When constructing a packet, each protocol prefixes its control information to the packet and passes the resulting packet to the next lower layer. When a packet is received, the lowest layer inspects and removes its header and hands the resulting packet up to the next higher layer.

This packet-switched network paradigm is fundamentally different from the telecommunication networks upon which it is physically built. Telecommunication networks are circuit-switched, in which a dedicated connection is sets up between the two endsystems for their exclusive use for the duration of the communication. The packet-switched Internet, in contrast, is connectionless in that no such connections are established; this enables it to optimize the use of the channel capacity available in a network, to minimize the transmission latency (i.e. the time it takes for data to pass across the network),

and to improve the network's scalability.

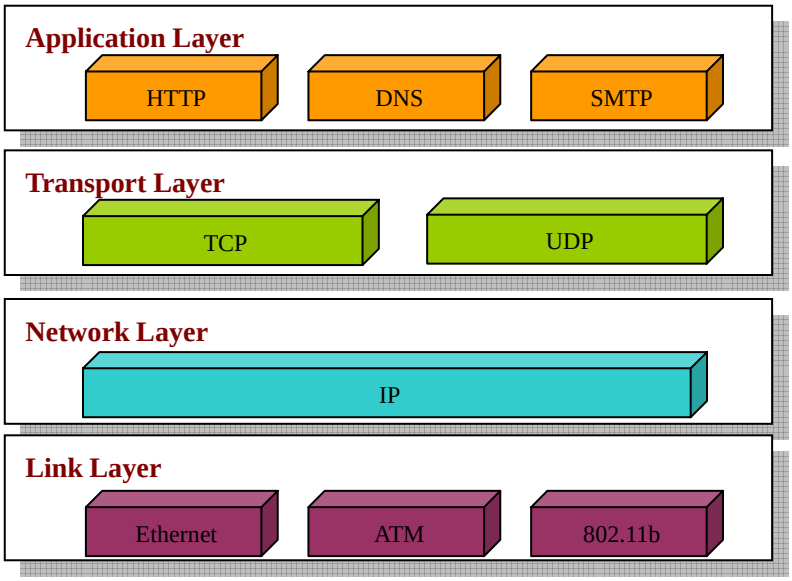


Figure 2-1: The layered protocol stack for the Internet

This connectionless nature of the Internet manifests itself as *statelessness* – *i.e.*, routers do not maintain any fine-grained information about traffic that they have routed. As a consequence, the Internet itself *is not able* to provide sufficient information to support network management activities. Thus, additional network measurement and monitoring must be provided to support network management.

2.1.2 Autonomous System (AS)

The Internet is comprised of a collection of separately managed networks. These networks are either operated as commercial services e.g. Internet Service Providers (ISP), or non-profit governmental organizations, educational institutions, or private companies. To enable network operators to collaborate so as to ensure the interconnection of each part of the Internet, as well as to maintain independent control of each part's resources, the network is organized into Autonomous Systems (ASes).

An Autonomous System is a collection of IP networks and routers, normally under one administrative authority, that presents a common routing policy to the Internet. A unique 16-bit AS number (or ASN) is allocated to each AS by the Internet Assigned Numbers

Authority (IANA)[1]. Depending on their connections and operations, ASes can be grouped into three categories: multi-homed ASes maintain multiple (redundant) connections to other ASes; stub ASes only maintain one connection to one other AS; and transit ASes provide connections through themselves to separate networks.

The traffic between ASes is exchanged at connection points, i.e. *peering routers* or *exchange points*; the connection is realized by establishing a link between routers in each AS. Routers connecting two ASes are called *core routers*, while routers connecting the AS with its customers (e.g. either residential customers via dial up lines, or business customer via higher bandwidth connections) are called *access routers, gateway routers,* or *edge routers* for being at the edge of network.

2.1.3 Intra-AS vs. Inter-AS Routing

Once a packet is placed into the network by an endsystem, it must be forwarded toward its eventual destination as specified by the packet's destination IP address. Routing is the activity conducted by a router to make the decision of which outgoing interface should be chosen for the packet and how to do so [Crovella06].

In the Internet, routing is hierarchical with two levels: intra-AS routing (also known as *intra-domain*) and inter-AS (also known as *inter-domain*) routing. Within an AS, normally, routers use the same Interior Gateway Protocol (IGP), although different ASes may use different IGPs. Between ASes, routing is achieved using the Border Gateway Protocol (BGP). The benefit of using such hierarchical routing is that it improves routing scalability in the sense that network state changes only impact a limited number of routers, and it allows engineering independence among autonomous systems.

Intra-domain Routing The purpose of an intra-domain routing protocol (IGP) is to exchange information among routers within a domain therefore to corporately manage communication among local networks. Interior gateway protocols can be divided into two categories: distance-vector routing protocols and link-state routing protocols.

In distance-vector routing protocols, each router does not possess information about the full network topology. It advertises its distances from other routers and receives similar advertisements from other routers. Using these routing advertisements each router

[1] From 1 January 2007, applications that specifically request 32-bit AS Numbers have been processed by the RIPE Network Coordinate Centre [RIPEWeb] [Kühne06].

populates its routing table; in the next advertisement cycle, a router advertises updated information from its routing table. This process continues until the routing tables of each router converge to stable values. This set of protocols has the disadvantage of slow convergence; however, they are usually simple to operate and are well suited for use with small networks. Some examples of distance-vector routing protocols are the Routing Information Protocol (RIP) and the Interior Gateway Routing Protocol (IGRP).

In the case of link-state routing protocols, each node possesses information about the complete network topology. Each node then independently calculates the best next hop from it to every possible destination in the network using its local topology information. The collection of best next hops forms the routing table for the node. This contrasts with distance-vector routing protocols, which work by having each node share its routing table with its neighbours. In a link-state protocol, the only information passed between the nodes is information used to construct the connectivity map. Some examples of link-state routing protocols are the Open Shortest Path First protocol (OSPF) and the Intermediate System to Intermediate System protocol (IS-IS).

Inter-domain Routing Inter-domain routing affects the exchange of traffic between ASes, therefore economic factors are involved. When an ISP connects two or more other ISPs, and each of them can carry traffic toward a particular destination, the choice of which one to use depends on business agreements covering organization-specific commercial reasons, i.e. policy routing. BGP is the protocol used for inter-domain routing, which has the ability to enable efficient inter-domain routing, while at the same time preserving the ability of individual organizations to employ policy routing.

BGP has this ability as it does not use traditional IGP metrics, but makes routing decisions based on paths, network policies and/or rule sets. BGP binds together the concepts of network address blocks and autonomous systems into a path vector based routing approach. A path vector is a particular network ID (or a collection of network IDs) and the sequence of ASes along the path to that network. Thus a path vector does not indicate the precise path a packet should follow within an AS, nor does it maintain a complete map of the topology of the Internet on a link-by-link basis. Rather, it is a specification of a particular AS path that can be taken to reach the given network. In other words, BGP uses a level of abstraction that views the Internet as a set of routing domains and maintains a routing map of the network at this AS level [Crovella06].

2.2 Network Measurement and Monitoring

This section discusses network measurement and monitoring. It firstly defines a network management loop and outlines the importance of measurement and monitoring in this loop. Then it introduces the focus for each of them in this context. Note that, in the general case, it might not be necessary to explicitly distinguish network measurement from network monitoring. The main purpose of discussing them here is to give prominence to this dissertation's focus, thus the problem to be solved can be better understood. As a case study, the Simple Network Management Protocol (SNMP) is introduced, showing how network monitoring facilitates network management activities.

2.2.1 Roles in Network Performance Management

By "network performance management" is meant the discipline of optimizing how networks function - i.e. attempting to deliver lowest latency, highest capacity, and maximum reliability despite intermittent failures and limited bandwidth [CiscoMng].

In the early days of the Internet, *network management* was nearly synonymous with a *network administrator*. Today, *network management* can be taken as an all-inclusive concept, including a wide variety of techniques such as traffic measurement, network monitoring, and QoS-based traffic engineering; in addition, the boundary between network measurement and monitoring is not normally explicitly distinguished.

To give prominence to this dissertation's focus, general network management can be decomposed into three related sub-steps, namely, measurement, monitoring and management. As illustrated in Figure 2-2, their foci are slightly different.

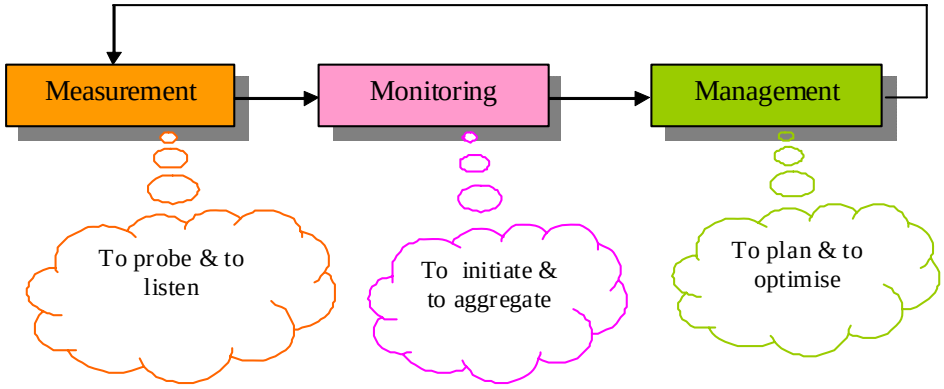


Figure 2-2: The relationship between measurement, monitoring and management in a network management loop

For network measurement, the focus is to perform the traffic measuring task by measuring the amount and type of traffic on a particular network, and recording the raw measurement results. The essential concern is how to deal with imperfect measurement devices and tools: the limitations of devices and tools i.e. the precision, will impact their selection, and the evaluation of the measurement result (i.e. errors may occur during the application of a particular tool, therefore calibration is needed to detect, and sometimes to correct, such errors).

For network monitoring, the focus is to initiate measurement tasks, collect raw measurement results, and report aggregated outcomes. The essential concern is how to obtain useful monitoring information at minimum cost; in other words, a system structure that guarantees components such as data aggregation, data repository, and data analysis can operate efficiently and interact seamlessly. In addition, an appropriate level of granularity for observation must be chosen.

Finally for management, the focus is to plan, allocate and optimize network resources based on the outcomes from the other two steps; the resulting strategy might lead to another measurement task being initiated. The essential concern for network management is how to maximize the overall performance, in terms of capability, latency, and reliability, given that the network has to carry traffic for different applications, and different applications introduce different requirements and impacts on network performance.

In this context, network measurement and monitoring are the fundamental steps in the network management control loop, in the sense that they provide reliable information and accurate analysis in characterizing and troubleshooting a network's performance.

Furthermore, compared with network measurement, network monitoring is more important, since the testing environment that network monitoring provides is required for particular network measurements to be conducted.

2.2.2 Measurement: Entities, Metrics, Methods

As discussed, the focus for network measurement is to perform traffic measuring tasks; it can be depicted from three aspects: the physical entities that are measured, the performance metrics that describe the measurement, and the methods that are used for measurement. These three essential elements are introduced in the following sections.

2.2.2.1 Entities

In the context of network measurement and monitoring, network entities are defined as follows [Lowekamp04]: "As networks are best represented in graphs formed with nodes and edges, network entities are divided into *nodes* and *paths*".[2]

A network node is not restricted to a single physical entity, but also can represent a range of devices including an autonomous system, a switch or a virtual node. Network nodes are generally classified into hosts and internal nodes; the latter stands for entities capable of forwarding traffic, and can be routers, switches, proxies, as well as more general concepts such as autonomous systems or virtual nodes[3].

A path is a unidirectional connection from one node to another and is represented by ordered pairs of endpoints. Special attention should be paid that: 1) the type of path is distinguished by the type of nodes that the path attaches - nodes are distinguished as host, router, switch, etc; 2) each path can be divided into constituent hops – a hop is a subclass of path; the only reason for including a separate type of hop is that a particular measurement system might wish to include additional attributes for a hop that are not used for a path.

2.2.2.2 Metrics

Performance metrics are characteristics of packet-switched networks that affect end-user application traffic. In other words, a performance metric is a primary property of the

[2] Wireless networks have been steadily growing over the last several years. Discussion of relevant wireless technologies is out of scope for this dissertation.

[3] Virtual nodes are used to describe additional functionalities that might be found in a physical node.

network, or of the traffic on it [Lowekamp04]. In general, there are four groups of network characteristics that attract most researchers' attention: packet delay, packet loss, throughput, and packet jitter.

Packet Delay

Packet delay is generally defined as the interval between the time when the first part of an object passes an observation position, and the time when the last part of that object or a related object passes a second observation position [Paxson96].

Packet delay is the result of summing up various delays that a packet experiences when it passes through a network. Among them, *routing delay* is spent inside a router and can be further broken down to *packet processing delay* and *queuing delay*; *transmission delay* is experienced by a router when putting a packet onto a link; and *propagation delay* is time spent by a packet on the link. Packet delay can either be measured as *one-way delay* or *roundtrip delay*. The former is better in reflecting the inherent asymmetry in typical packet networks; while the later is easier to achieve, since one does not have to synchronize clocks, but that the asymmetric information, which may be crucial for management purposes, is eliminated.

Packet delay is important to measure since erratic variation in delay makes real-time application difficult to support, and exorbitantly low delay can indicate the path is lightly loaded.

Packet Loss

Packet loss is generally defined as: packets, sent out by a sender along a network path, are lost in transit, and, as a consequence, they are not received by their destination [Paxson96].

The most significant cause for packet loss is network congestion: when the output link of a router is busy, the router may explicitly drop packets destined for that output link. Since precise information regarding which packets are lost is difficult to obtain, and repeated loss may have a more severe impact than the loss of a single packet, statistical loss, l*oss average*, is most commonly used to measure packet loss. Loss average is defined as the average of the singleton loss values over a series of sent packets (given as a percentage between 0% and 100%) [Lowekamp04].

Packet loss is an important performance metric to measure since packet loss impacts the quality of service provided by network applications. The sensitivity to loss of individual packets, the frequency and patterns of loss among longer packet sequences, is strongly

dependent on the application itself. For streaming media (audio/video), packet loss results in reduced quality of sound and image; for data transfer, packet loss can cause severe degradation of achievable bandwidth.

Throughput

Throughput is generally defined as the maximum amount of data per time unit that a hop or a path can provide given current utilization [Paxson96]. The maximum throughput of a node or communication link is synonymous to its *capacity*. The portion of capacity that is not being used during a given time interval is *available bandwidth*.

It is important to specify at which layer the throughput is being measured. Starting from the physical medium, throughput stands for the physical bit rate, whereas going up the protocol stack, the throughput apparently decreases by taking into account the framing at the Link Layer, and the protocol overheads at the Network and Transport Layer [Paxson96]. Throughput over a sequence of hops is determined by the elements(s) with minimum available capacity.

Throughput is obviously important in measuring the capacity and utilization of link resources.

Packet Jitter

Packet jitter, also referred to as *packet delay variation*, is generally defined for packets inside a stream of packets: by going through two measurement points, jitter is the difference between one-way-delay of a selected pair of packets [Paxson96].

Packet jitter is often measured by observing the inter-arrival times of packets sent at a fixed interval. Since it is the difference in delays between packets pairs, clocks do not need to be carefully synchronized.

Jitter is important in measuring the smoothness of a packet arrival process; it can reflect the queuing delays in routers along a path. Packet arrivals with low jitter are more predictable and often lead to more reliable application–level performance. [Crovella06]

Standardisation of Metrics

It is useful to point out that in network measurement, different metrics can be defined for different things in different contexts; they can be used to describe directly measured data; they can also be used to represent the derived values. For long term benefit, wide diversity in defining metrics will cause ambiguity in measurement infrastructures, and prevent the

interoperation of different manufacturers' products. Hence the Internet Engineering Task Force (IETF) has defined a standard set of network metrics, the IP Performance Metrics (IPPM).

IPPM consists of a series of RFCs in which an extensible and reusable framework for conducting such measurements is first defined (RFC2330), followed by the definition of four metrics: unidirectional-connectivity (RFC 2678), one-way packet delay (RFC 2679), one-way packet loss (RFC 2680) and packet delay jitter (RFC 3393); the series also define the three notions of *singleton*, *sample* and *statistics*, so that the measurement results can be well understood and evaluated. In addition, a few important notions, such as *"wire-time"* and *"packets of type P"* are also defined for the first time. Although, at the time of writing this dissertation, IPPM still does not define bandwidth-relevant metrics, it is a meaningful attempt at unifying metric definitions for network measurement.

2.2.2.3 Methods

Measurement methods in general fall into two groups: active and passive. In addition, in-line measurement has been proposed as a new measurement technique that provides an un-intrusive universal instrumentation mechanism for the next generation Internet by exploiting IPv6 extensibility mechanisms. In this section, each is discussed.

Active Measurement

Active measurement relies on injecting probe traffic with known characteristics into the network to test particular attributes of a network [Paxson96]. The purpose of probe packets is to provide some insight into the way that real network traffic is treated within the network.

Typically, an active measurement involves two measurement sites to send and receive probe packets respectively. Techniques adopting active measurement can measure the properties of end-to-end network paths between instrumented systems - by injecting probe packets at one end and retrieving the probe packets at the other end.

Commonly used active measurement tools include: 1) *ping*: send an ICMP ECHO packet to a target and capture the ECHO *reply* packet to check connectivity to the target, and to measure the instantaneous RTT between the sender and the target; and 2) *traceroute:* send a sequence of UDP probe packets with increasing TTL to an unlikely port on the destination target; by receiving the ICMP *time exceeded* packet, identify the intermediate nodes along the path; by receiving the ICMP *port unreachable* packet, identify the

destination target.

There are four advantages to active measurement approaches: 1) measurements can be conducted in a timely manner; 2) measurements can be loosely coupled with the underlying infrastructure and are close to end-user experiences; 3) measurements are not affected by constraints such as access control, privacy and security problems; and 4) measurements can be done when measurements can not be performed passively.

The main drawbacks with active measurement approaches include: 1) accuracy: the synthetic injected traffic does not necessarily reflect the behaviour experienced by real-world traffic: for example, injected traffic using different protocols, such as ICMP and UDP, may be processed and routed differently from that experienced by real operational traffic; 2) network impact: synthetic injected traffic causes extra traffic load and may impact the network, especially when the methodology is not carefully designed to minimize the amount of the synthetic traffic [Paxson96].

Passive Measurement

Passive measurement does not inject any measurement traffic into the targeted system; it typically implements packet filters to capture initial network traffic and then searches for particular events using pattern-matching techniques [Hegering99]. The common API for packet capture is *libpcap*, which is available on most operating systems; it provides entry points for specifying the interfaces to be monitored and the types of packets to be collected. After the *libpcap* library delivers raw packet data to interested applications, the application software can perform further processing, such as parsing of packet header fields and interpretation of network protocols.

Commonly used passive measurement tools include: 1) *tcpdump*[TcpdumpWeb] (windump for the Windows platform) which collects packets transmitted and received by systems running Unix/Linux operating systems; and 2) *NetFlow* [NeflowWeb], which is a monitoring system available on Cisco routers, collecting flow statistics observed by a network interface.

The advantages of passive measurement include: 1) no extra traffic: extra injected traffic load is eliminated, and any consequent biases on the resulting analysis are eliminated; 2) easy to deploy: typically, a passive measurement is performed at one monitoring site thus no extra cooperating sites are required; and 3) suitable for special measurements: for example, inter-AS routing can be passively measured between two BGP routers when those two ASes (i.e. not the complete Internet) are the measurement targets.

The main drawbacks with passive measurement include: 1) significant data volume: as line rates continue to increase, the amount of network traffic at the monitored links is substantial; 2) difficult data access: routers at which important measurement data are stored are usually backbone routers or controlled by ISPs, making it harder to access; and 3) complicated data processing: since the captured measurement data is asynchronous with real-time traffic, it is difficult to correlate samples collected at two distinct observation points to yield one-way flow measures, and the correlation consumes significant resources and network bandwidth [Fraleigh01].

In-line Measurement

The in-line measurement technique may be viewed as a hybrid of the beneficial characteristics of active and passive measurement approaches, whereby the measurement data and triggering mechanisms are piggybacked onto real user packets [Pezaros04]. To do so, it exploits IPv6 extension headers to instrument portions of the network traffic, in order to guarantee that the measurements reflect the service experienced by real user traffic.

The advantages of in-line measurement include: 1) low overhead: with low overhead and minimal impact on network traffic, in-line measurement provides a high-level of probability that the real user experience is being measured; it is equally applicable to measuring aggregate flows as it is to particular applications or protocols; 2) dynamic deployment: it is well-suited to dynamic deployment and presents a scalable implementation; 3) service-oriented approach: by adopting a service-oriented approach, in-line measurement can form the basis of end-to-end measurement instrumentation that is able to reveal the effects of network behaviour on traffic flows; and 4) correlation is not needed: by avoiding the need for correlation of measurement samples from numerous measurement sites, the transfer of measurement data is kept to a minimum level; consumption of network and computational resources can therefore be reduced.

The drawbacks and tradeoffs of using in-line measurement include: 1) extension headers need to be defined and encapsulated in the link level frame; if the length of application payload and all of the headers extends the length of a link level frame, fragmentation of network level packets may occur; 2) filtering and sampling mechanisms need to be exploited to address scalability and overhead issues, and the applicability of in-line measurements for particular application domains and network operations, is not fully exploited.

2.2.3 Monitoring: Location, Clock, Data

Compared with the basic measurement concepts outlined in the last section, this section is more about how to design and initiate network measurements; in other words, practical issues on how to organize network measurements are discussed, such as where a measurement can be made (location), what is the role of time in network measurement (clock), and what to do with resulting measurement data.

2.2.3.1 Location

The first practical issue of designing a sound measurement is to choose the right location (also referred to as the *measurement point*) at which the measurement can be performed. It can be looked at from different perspectives – from the number of locations that a normal method needs, from the physical location in a network, and from the size of the network that being measured.

Single-Point vs. Multiple-Point

Regarding the number of locations involved in a measurement, passive measurement normally is conducted at only one location where the operational traffic can be observed and recorded. Active measurement involves at least two locations, i.e. probe sender and probe receiver; but when a single probe is multicast so that the performance characteristics can be inferred from the correlation of a set of interacting network paths, multiple locations are involved.

Various Locations within Physical Networks

Some locations are more appropriate than others to measure certain metrics: 1) local area networks are good for measuring endsystem-relevant metrics, such as local latency, and hardware-related metrics; 2) at the entry points into a network, metrics such as access control, overall flow statistics, the fraction of traffic destined to customers, and the portion of traffic that is transmitting through the network, can be measured at gateway routers; metrics such as inter-domain connectivity, convergence of inter-domain routing, locating of route loops etc, can be measured at peering routers; customer relevant measurement, such as measuring failure rate in service availability and assurance in service level agreement, can be conducted at access routers; 3) inside a backbone network, measurements are conducted largely from an intra-organizational point of view: to properly anticipate the provision and the utilization of the resources; to provide timely indication of required

updates tp routers, links, and protocols; and to obtain a detailed view of current routing information.

End-to-End vs. Edge-to-Edge

When a measurement is performed in a co-ordinated fashion involving multiple locations, depending on the scale that a measurement covers, it normally falls into two categories: end-to-end and edge-to-edge. [4]

End-to-End measurement refers to measurements being conducted on a collection of endsystems; therefore, it is possibly measuring a path that crosses several network domains. The benefits of conducting measurements on end hosts include: administrative access to intermediate routers is not required; the CPU cycles consumed on routers is split and shared by end hosts; and the measurement results reflect the network performance that user traffic experiences. However there is also a drawback: to obtain network-wide performance status, individual results from the participant end hosts have to be correlated and aggregated by using statistical methods, which are complicated, and the estimated result may not be as accurate as router-based approaches.

Edge-to-Edge measurement refers to measurement being conducted on a set of edge routers within a single administrative domain. Edge routers here refer to the ingress and egress points for the network, as apposed to core routers that switch the traffic among the routers of the network. Edge routers play an important role in the provision of Quality of Service (QoS) in the Internet. The Differentiated Services (DiffServ) [Habib04] architecture classifies and marks packets at edge routers so that the packets can be handled properly by scheduling and buffering algorithms in the core network to achieve their QoS requirements. Edge routers can also be used to test if QoS is satisfied, since edge-to-edge measurement can accurately reflect the performance status of a single administrative network in a fully controlled style. For example, to measure one-way delay, active measurements can be conducted in which synthetic probes can be injected from an ingress point toward an egress point over which normal traffic travels. However, the primary concern is to correlate and aggregate measurement results; and this type of measurement should be minimized due to the processor cycles and bandwidth consumed by the injected probes.

[4] An “end-to-end” path can actually be divided into three segments: two “end-to-edge” segments and a “edge-to-edge” segment.

2.2.3.2 Clock

Time plays an indispensable role in network measurement - almost all calculation of performance metrics requires the use of time information; and for multi-site correlated measurements, time is normally required to be synchronized. This section firstly introduces basic clock terminologies; then approaches for setting up synchronized clocks are introduced.

Clock Terminology

- **Resolution**: a clock's resolution is the smallest unit by which the clock's time is updated. It gives a lower bound on the clock's uncertainty.
- **Offset**: a clock's offset is defined as the difference between the time reported by the clock and the true time defined by national standards at a particular moment.
- **Skew**: a clock's skew is the frequency difference (first derivative of its offset with respect to true time) between the clock and national standards at a particular moment.
- **Accuracy:** a clock is accurate if the clock's offset is zero at a particular moment, and, more generally, a clock's accuracy is how close the absolute value of the offset is to zero.

Synchronization Approach

Synchronized clocks are required for many multi-site measurement tasks – for example, one-way packet delay involves recording the departure time of a packet at one location, and the arrival time of the packet at another location. To obtain accurate measurement results, either the clocks at each site are synchronized, such that the offset between each other is as small as possible; or the offsets and their effects on measurement results can be inferred and removed after the measurement.

For synchronizing multiple sites' clocks, the Network Time Protocol (NTP) [NTPWeb] is widely accepted and used for clock synchronization. NTP is organized into a hierarchy consisting of servers and clients. One level in the hierarchy is called a *stratum*; stratum 1 is synchronized to national standards by radio, satellite, or modem. A client normally tries to synchronize with multiple redundant servers over different network paths to obtain accurate and reliable time estimates. The calculated offset estimates from each server are then weighed and combined to produce a correction for the local clock.

For inferring the offset after a measurement is performed, it is compromised when tightly

synchronized clocks are unavailable – the problem can be thought as inferring the *relative offset* and *relative skew* of the two clocks involved. In the case of relative delay, it is acceptable and sometimes tolerable, since in general, one-way delay measurement is often on the order of tens or hundreds of milliseconds, which gives an approximate range of the accuracy needed in relative delay estimation. In the case of relative skew, it is more challenging and important since many measurements involve *variation* in delays therefore relative skew does affect measured delay variation. Some solutions are proposed to remove the relative skew, such as by fitting median lines to the data [Paxson98], by linear programming [Moon99], or by computation of convex hulls [Zhang02].

2.2.3.3 Data

For network measurement and monitoring, data is one of the most important elements. *Data* here mainly refers to characteristics of the network traffic that are of interest and the measurement results that are to be processed. This section discusses data from these two perspectives.

Capturing Data

The question of how data are captured can be discussed along two axes: vertically, data can be captured at different granularities i.e. different layers of the protocol stack; horizontally, data can gathered in different infrastructures, i.e. locally, remotely, or in a distributed style. A suitable infrastructure selected for a measurement task is mainly affected by the granularity of data. Thus, the following discussing on data capture is along the vertical axis, that is, according to the bottom-up sequence, the layers where data can be captured are divided into three levels: the router- and link- level, the transport level, and the application level.

- Router- and Link- Level

  Raw packets are captured at router- and link- level, where large amounts of data are processed quickly to avoid packet delay or loss. Resources at this level are too tight to be reserved for specific tasks due to unexpected bursts and fluctuation of traffic. Thus, capturing data at this level, even for simple metrics such as packet counts, is consuming significant resources.

  At this level, data is typically only suitable to be gathered locally, i.e. in an Autonomous System. This is due to the fact that the data at this level are captured from routers and links, which are normally under strict administrative control. Therefore it is

virtually impossible to be triggered in a remote or distributed style.

- Transport Level

The volume of data remains large and the ability to process any reasonable fraction of data in real time is difficult; hardware monitors often must be deployed to handle high-speed links. Even so, it is still extremely hard to pick out the information that is of interest in the first place; so the full size of data is captured, followed with a filtering process being applied to obtain the desired information. This information can be used to aggregate IP packets into IP flows. An IP flow is defined as a set of packets distinguished by their source and destination addresses, or any other function of their IP or transport fields [Crovella06]. The most widely used fields to identify an IP flow are the source and destination IP addresses and port numbers, protocol, and the start and end time of flow; this is opposed to a network flow which is defined based on ingress and egress routers.

Flow level data are best gathered locally. They, too, cannot be remotely triggered as that requires administrative level access to the monitored link. It is challenging if a distributed infrastructure is selected where multiple packet streams have to be captured simultaneously, as a variety of clock skew issues have to be handled, unrelated packet streams have to be separated out, and the streams of interest have to be reconstructed.

- Application Level

At application level, capturing data is generally easier, in the sense that the time constraints are human awareness timeframes (rather than at microsecond rate as lower levels require); and logging can be readily used and configured, catering for different requirements.

However, this wide space of configuration leads to significant diversity in data, therefore the configuration and logging software, which can be written in scripts and programs, must be tailored to various usage scenarios.

The generally better and more widespread facilities, such as higher level software, commands, and control structures, enable application level data being gathered locally, remotely, and at distributed locations, which makes distributed installation, initiation and suspension easier. The difficulties with clock skew and coordination of simultaneous collection are still present, but less problematic than for lower-level data gathering.

Processing Data

Once data are captured and gathered, one must manage and use the data to manage the network. As this can be very complicated, involving several theoretical and practical orientations, a detailed and deeper discussion of data processing is outside the scope of this dissertation; some important issues and solutions are highlighted here.

- Management

Firstly, data should be well *managed* through the construction and use of specialized tools. The common thread among these tools should be using sophisticated algorithms to operate on large data sets. This can be achieved either through careful software design at the packet capturing stage [Paisley06]; or through *stream database* technology given the special characteristics exhibited by traffic data [Cranor03]).

- Reduction

Secondly, the volume of data should be *reduced*; otherwise the immense size can make subsequent analysis difficult. Here traffic data reduction, to some extent, can be envisioned as lossy compression with specific traffic features maintained. The most common methods in data reduction are the use of traffic counters provided by SNMP, and the collection of flow records. More sophisticated methods include sampling-based approaches, sketch-based summarization, dimensionality reduction, and probabilistic models [Crovella06].

- Analysis

Finally, the data should be *analysed* to obtain hidden information valuable for network management. *Hidden information* means either the information can't be seen from a single packet trace, e.g. topology information; or the information that is not publicly available out of administrative concern, e.g. internal measurement results from ISPs, or the information has to be deduced by statistical processing, e.g. traffic pattern of a particular application.

Broadly speaking, traffic analysis for large-scale networks involves inferring network infrastructure or performance parameters based on a limited subset of the captured measurement data. In addition, a variety of assumptions have to be made, e.g. the links can be treated as unidirectional or bidirectional; paths are symmetric or asymmetric, round-trip or one-way. Furthermore, statistical models must be constructed for validation study and feasibility analysis.

2.2.4 Case Study: Simple Network Management Protocol (SNMP)

As a case study, this section introduces the Simple Network Management Protocol (SNMP) to show how network monitoring facilitates network management activities. The contents that are covered in this section include SNMP's high level infrastructure and evolution history, the information management mechanism, and the SNMP-based network monitoring functionalities.

2.2.4.1 Overview

As the *de facto* standard, SNMP "is used by network management systems to monitor network-attached devices for conditions that warrant administrative attention" [SNMP98].

An SNMP-managed network consists of three key components, namely SNMP agents, managed devices, and network-management systems (NMSs): a SNMP agent is a network-management software module that resides in a managed device; a managed device is a network node residing on a managed network and capable of communicating status information to the outside world by running a SNMP agent; and a NMS is a general-purpose computer running special management software which communicates with the SNMP agents over a network, issuing commands and getting responses.

The basic relationship between the three components is as follows: managed devices can be routers, switches, bridges, hubs, computer hosts, or printers, as long as it is enabled to collect and store management information and make this information available to NMSs by using SNMP; a SNMP agent has local knowledge of management information and translates that information into a form compatible with SNMP; and NMSs provide the bulk of the processing and memory resources required for network management. "One or more NMSs can exist on any managed network" [SNMP98]. The polling approach, i.e. NMSs periodically query managed devices for information, can indicate whether or not each network element is operating within the configured operational parameters, and alert the network operator when there are local anomalies to this condition.

The initial implementation of the SNMP protocol (RFC1157) was published in May 1990. The second version, SNMPv2 (RFC1448) was published in April 1993 to address some limitations imposed by the first version of the protocol, such as the inefficient transmission of large amount of management data, the manager-centric polling scheme, and the use of the connectionless UDP transport for the encapsulation of SNMP messages. However

SNMP Version 2 was not widely adopted due to serious disagreements over the security framework in the standard and is incompatible with SNMPv1 in two key areas: message formats and protocol operations. A third version SNMPv3 (RFC 3411) was published in December 2002 defining an architecture that will enable extensibility and minimal implementations, at the same time support more complicated features required in large networks.

2.2.4.2 Management Information Base (MIB)

Broadly speaking, in SNMP, NSMs communicate with agents though a *Management Information Base (MIB)*, which is a collection of information that is organized hierarchically and is comprised of managed objects identified by object identifiers.

To identify and describe these managed objects, a rootless global identifier tree was introduced by the International Organization for Standardization (ISO) and International Telecommunication Union, Telecommunication Standardization Sector (ITU-T). In the global identifier tree, each object is assigned a unique worldwide identifier matching to a tree node; each tree node is assigned a name and a number beginning with 1 and enumerating all nodes belonging to the same parent node. This hierarchical model permits management across all layers of the OSI reference model to extend into applications such as databases and e-mail, as MIBs can be defined for all such area-specific information and operations.

In the identifier tree, the levels are assigned by different organizations: the top-level MIB object IDs belong to different standards organizations, while lower-level object IDs are allocated by associated organizations. Vendors can define private branches that include managed objects for their own products. Regarding the meaning that each tree node presents, the internal nodes of the tree are only used for registration and object identification; while the actual management information is located only in the leaves of the tree, whose instantiation produces the actual managed objects in the agent MIB.

To enable NMSs to access the remote agenst's MIBs, such as to retrieve or set the variables in the agent MIB, the SNMP protocol defines a standardized set of operations, e.g. *Get*, *GetNext*, *Set*, and *Trap*. These operations are implemented through exchanging messages in a request/response style, i.e. the NMS issues one of the four requests, and managed devices return the response.

2.2.4.3 SNMP-based Network Monitoring: RMON and RMON2

Remote Monitoring (RMON) [RMONWeb] is a standard monitoring specification that enables various network monitors and console systems to exchange network-monitoring data. RMON provides network administrators with more freedom in selecting network-monitoring probes and consoles with features that meet their particular networking needs.

The RMON specification defines a set of functions that can be exchanged between RMON-compliant console managers and network probes. The RMON MIB enables a probe to perform diagnostics, as well as collect performance, fault and configuration information without continuous communication with NMSs. The monitor can log and store performance information; this performance information can then be played back by the NMS in order to obtain historical knowledge and perform further empirical diagnosis into the cause of a problem (RFC2819). As such, RMON provides network administrators with comprehensive network-fault diagnosis, planning, and performance-tuning facilities.

The first version of the RMON specification was published in February 1995 (RFC 1757), which was mainly designed for Ethernet networks. In January 1997, the RMON2 MIB specification (RFC 2021) was published; it extended the original RMON MIB to enable the RMON MIB to monitor protocol traffic above the media access control (MAC) level. This extension overall improves network monitoring since more information processing takes place in the RMON agent (remote system), and the SNMP data transfer from the agent to the manager is reduced [Hegering99].

RMON and RMON2 are considerably more advanced than the rest of the SNMP MIBs, since they carry out network monitoring data pre-processing and statistical computations at the agent, and can hence minimise the data exchange between agents and management stations. RMON agents can also be concurrently communicating monitoring data and statistics to multiple managers responsible for different units and/or functions within the network.

However, RMON’s and RMON2’s advantages come at the expense of complicated table management and the need for complex agents to be implemented in dedicated devices, therefore a number of network components only partially integrate RMON agents [Hegering99].

2.3 Overlay Networks

This section provides background knowledge on overlay networks. It starts with overlay

network definition and a brief introduction to the evolution of overlay networks. Next, a case study follows: as a representative overlay network, as well as the basis for a number of other overlay networks and applications, Peer-to-Peer (P2P) networks are discussed. Regarding related work that is of particular relevance to this work, that discussion is deferred to Chapter 3, where the foci are on the technology they employ and their influence on this work.

2.3.1 Fundamental Concepts

An overlay network is a virtual network built on top of a physical network. Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path through one or many physical links along the underlying network.

The idea of overlay networks is not a new one. Usenet News was, perhaps, the oldest application layer overlay network that implemented decentralized control with relatively simple administration to allow users to post messages to a newsgroup; it was originally based on a protocol called UUCP (Unix-to-Unix-copy protocol) in which one UNIX machine could automatically dial another, exchange files with it, and disconnect.

As the Internet has grown in popularity, many modern overlay networks are built using the Internet as the underlying network. Such applications mainly fall into two groups: 1) to directly provide application-level functionality that is out-of-scope for the underlying network, e.g. Content Delivery Networks (CDNs) [Vakali03] and Resilient Overlay Network (RON) [David01]; and 2) to alleviate the effects of slow or sporadic deployment of new services in the Internet, e.g. application-level multicast [El-Sayed03].

The ever growing popularity of building overlay networks on the Internet reflects the Internet's design principles. One of the original principles of the Internet's design is to provide *connectivity* among a wide variety of computing devices that use a wide variety of communicating technologies [Crovella06]. To achieve this, 1) IP is designed as the only Internet protocol, so that for a new communication technology, it is only necessary to find a way for the technology to transport IP packets; 2) the switching elements (routers) are designed to simply inspect the packet's IP header to determine the outgoing link, without examining headers from any higher-level protocols; 3) following the end-to-end argument [Saltzer84], endsystems are required to complete network connections, hence to offload that complexity from routers which can then focus on delivering packets for network connections.

Given the state of the art of single-chip technology, as well as the relatively low data rates and small number of network flows at network edges, gateway routers, access routers, or even end hosts are able to perform multiple functions to meet new application demands cost-effectively. Overlay networks built upon the Internet at the network edge demonstrate this feasibility, and are now being considered in many application domains, such as content distribution, conferencing, gaming, and network measurement and monitoring.

2.3.2 Case Study: Peer-to-Peer Networks

As a case study, this section introduces P2P networks, which are representative overlay networks, as well as the basis for a number of other overlay networks and applications. The contents that are covered in this section include P2P's emerging background, its adaptation to the current Internet infrastructure, and its primary routing paradigms.

2.3.2.1 Overview

The P2P model of network communication can be traced back to the initial design of the ARPANET in the late 1960s. Early successful P2P applications include Usenet News and DNS (Domain Name Server) [Minar01]

Contemporary P2P networks have grown dramatically since the mid-1990s, primarily to sidestep the restrictions caused by firewalls, DHCP (Dynamic Host Configuration Protocol) and NAT (Network Address Translation), which in common make access to the Internet more controlled and managed. Napster [NapsterWeb] was the first modern P2P application designed for MP3 file sharing. It differentiated the concepts of information ownership from the traditional client/server paradigm (where clients can only download files from servers), and allowed many Internet users to distribute information in a free and pseudo-anonymous manner [Minar01].

Following Napster, a large number of P2P file sharing [BitTorrentWeb, GnutellaWeb, and eDonkeyWeb] and distributed storage [Zhao04, Ratnasamy01, Stoica01, and Rowstron01] applications emerged. The success of P2P applications in these fields has been the primary reason for the level of interest in P2P, and has encouraged adoption of P2P architectural concepts in other distributed application areas, such as streaming media delivery [Stolarz01], distributed collaboration [Cugola02], lightweight sensor networks [Triantafillou03], mobile ad hoc networks [Kortuem01], and data management [Huebsch03].

2.3.2.2 System Level Concepts

At system level, P2P network features can be summarised from three aspects:

- **A Communication Model with Minimal/No Dependence on Central Server(s)**

  In the client/server model, the performance of the whole network does not depend on the majority of end hosts, but on a minority of central servers - i.e. redundant servers are used to provide backup. Compared to the high volume of client requests, such processing capabilities are under-provisioned. In addition, the expense and heavy administration on servers make the situation worse.

  In the P2P model, network entities communicate with each other equally and directly, therefore the workload and responsibilities have been distributed. With minimal/no dependence on central servers, the P2P model eliminates central points of failure and server side bottlenecks [Minar01].

- **Resources from the Network Edge**

  A key element of the success of the P2P model is that it takes advantage of resources at the network edge, i.e. millions of end-user PCs world-wide, in contrast to the client/server model in which performance primarily relies upon a few servers and ignores the huge potential capability of clients.

  More importantly, the P2P model is consistent with today's commercial reality: the Internet has become a routine part of people's daily lives; end users are keen to be a part of it - not only to consume, but also to contribute.

- **Overlay Network Construction**

  P2P network construction depends upon five key elements [Gribble01]: 1) *content name* describes what a user is looking for, such as a file name in a file system; 2) *node address* describes where the network node is, for example, an IP address describes where a host resides in the Internet; 3) *routing mechanism* discovers or disseminates routing information; 4) *network topology* describes the set of physical or logical links between network nodes; and 5) *lookup function* binds content names and node addresses registered in the system.

  Among these five elements, content name and node address can be associated to answer the question of "Where is the content?" or "Where should the content be placed?", whereas routing mechanism, network topology, and lookup function are bound together to answer the question of "How to get there?" According to the answers of these

questions, P2P networks are classified into different categories, which are discussed in the following subsection.

2.3.2.3 P2P Routing Strategies

Broadly, according to their routing strategies, P2P networks can be classified as *structured* and *unstructured*; *epidemic* P2P networks are also recognized as a third category [Ayalvadi03]. In the case of unstructured P2P, for the degree of decentralization, it can be further divided into three groups: partially decentralized, purely decentralized, and hierarchical.

Structured P2P

In common, structured P2P systems [Zhao04, Ratnasamy01, Stoica01, and Rowstron01] employ distributed hash tables (DHT) to implement routing requests: the content is associated with a key produced by a hashing function. Each node in the system is responsible for storing a certain range of keys. The core operation is the key-word matching function *lookup(key)*, which efficiently routes the query requests to the node storing the object with that specific key.

Different DHT-based routing algorithms choose different underlying routing geometries, such as hyper-cubes [Ratnasamy01], ring [Stoica01], tree-like structures [Zhao04, Rowstron01]; these geometries differ in the algorithms' degree of resilience and proximity awareness.

Due to the nature of DHT, structured P2P systems are evenly distributed and scalable only at application level – in the sense that the hash functions discard the underlying network topology so that in a large scale network with high rate of churn (i.e. nodes join/leave frequently), it is hard to keep a satisfactory level of performance, e.g. in terms of bandwidth usage and packet delay.

Unstructured P2P

Compared with structured P2P, unstructured P2P systems are much easier to build and straightforward to understand. The overlay networks are constructed using application-level protocols or primitives (e.g. ping/pong), whereas the session of query/reply is constructed over transport-level protocols such as TCP and UDP.

Unstructured P2P systems are known to lack scalability. Therefore centralization is introduced to tackle this issue. Depending upon the degree to which a system relies on

centralization, there are three categories:

- Partially Decentralized (e.g. Napster [NapsterWeb], BitTorrent [BitTorrentWeb]) approach remains to have central server performing special roles such as index maintenance or membership management. A peer has to submit a query to this server for the location of the content it is looking for. Once the query is answered, file downloading is performed directly between source and destination peers.
- Purely Decentralized (e.g. Gnutella [Gnutella]) approach has no central server at all - all network peers act equally and symmetrically. Flooding algorithms are usually employed to route queries.
- Hierarchical (e.g. eDonkey [eDonkeyWeb]) approach has super nodes performing special functions. Super nodes normally have significant processing resources and good quality of network connections. They are dynamically appointed; once failure occurs, other candidate peers are promoted as super nodes.

Epidemic

Epidemic (e.g. Astrolabe [VanRenesse03], Lola [Rodrigues04]) algorithms disseminate information in a distributed system in the same way as an epidemic would be propagated throughout a group of individuals: each process of the system chooses random peers to whom it relays the information it has received. Another analogy is with the spread of a rumor among humans via gossiping (hence the algorithms are also called gossip algorithms).

These algorithms are descendants of more traditional flooding algorithms, where various means are taken to reduce the amount of flooding, such as membership management and message filtering. Epidemic algorithms are scalable in the sense of underlying peer-to-peer communication model between nodes - both membership management and gossip dissemination; epidemic algorithms are resilient in the sense that individual failure will not be fatal - the routing request will eventually be resolved successfully.

2.4 Summary

This chapter has provided background information required for this dissertation.

It starts from revisiting the architectural and operational concepts of the Internet. By pointing out the fundamental differences between the connection-oriented telecommunication network and connectionless Internet, it is clear that the Internet is

stateless – i.e., routers do not maintain any fine-grained information about traffic that has been routed. As a consequence, the Internet itself is not able to provide sufficient information to support network management activities. Thus, additional network measurement and monitoring must be provided to support network management. Then, through the introductions of how the Internet is organised and how packets are routed, the general environment for a network monitoring system to be deployed is presented, which paves the way for introducing more detailed knowledge on network measurement and monitoring.

Network measuring and monitoring are important for proper and efficient operation of a network. By decomposing their roles in a network management loop, the focus for each of them is explicitly pointed out. Thus, the emphasis of this dissertation, i.e. a network monitoring system focusing on system level initiation of measurement tasks, is prominent. Lastly, SNMP is introduced, as the *de facto s*tandard showing how network monitoring facilitates network management activities.

Finally, overlay networks are introduced: firstly with fundamental concepts; then a case study of Peer-to-Peer (P2P) networks, by summarising its emerging background, its adaptation to current Internet's infrastructure, and its taxonomy of routing strategies. The reason to cover P2P networks in this dissertation is that a broad range of overlay technologies are either based upon P2P networks, or influenced greatly by P2P networks.

As for related research that is of particular relevance to this dissertation, that discussion is deferred in the next chapter, once the problem to be solved in this dissertation is precisely defined.

Chapter 3

Problem Analysis

In this chapter, the problem that this dissertation is to address is critically analysed. It starts with answering the question of why an overlay-based network monitoring system is required, in Section 3.1. Next, in Section 3.2, OverMon, the overlay-based monitoring system that is proposed, is precisely defined from different perspectives. This is followed by a discussion of the overlay techniques that can potentially be applied in Section 3.3, and the related work that is of particular relevance to this dissertation in Section 3.4. Finally, Section 3.5 summarizes the whole chapter.

3.1 Why Overlay-based Network Monitoring?

This section discusses why an overlay-based network monitoring system is required. This question is answered by answering four sub-questions, namely what the new challenges are, what the weaknesses in conventional approaches are, how can overlays help, and at what cost.

3.1.1 What are the new challenges?

Following the rapid development of applications for the Internet, for example, the World Wide Web at first, then overlay networks subsequently, it is widely accepted that current network systems are continuously evolving towards 'next generation networks (NGNs)', in which the network behaviours are more difficult to manage. The new challenges that accompany this rapid evolution can be summarized as below [Liotta02]:

- Firstly, from the aspect of topology, compared to the old situation where network topologies are carefully planned and relatively static, current and future networks are expected to be dynamically constructed in real time. For instance, overlay networks, such as P2P file-sharing networks, can be dynamically constructed and

destroyed with the start and the end of a downloading session, which is fully controlled by users at the application level.

- Secondly, since a network can be easily constructed in the form of overlay networks, the size of a network service (i.e. in terms of the number of connected nodes) can grow dramatically. Hence, a next-generation network is characterized as highly distributed at a very large scale.
- Thirdly, as network infrastructures evolve, heterogeneous network technologies, services and applications coexist and interact; the variety of these factors along with the possibility of accessing them from virtually *any* location, make it extremely difficult to foresee the type and distribution of the network traffic.

3.1.2 What are the weaknesses in conventional approaches?

These challenges make current and future networks very difficult to manage and control, unless monitoring systems become more efficient. Unfortunately, conventional network monitoring systems have difficulty in addressing these issues:

- Static centralised monitoring systems, such as SNMP-based approaches [SNMP98], are unable to address the increasing topological dynamism in that measurement tasks and aggregation structures are deployed among a fixed number of agents and managers; thus, there is little scope for the monitoring system to detect and act upon topology changes. The frequency of SNMP-based polling needs to be sufficiently high to detect network topology changes; frequent polling causes high measurement data volumes and significant measurement overheads, leading to potential server side bottlenecks and single points of failure. They are unable to address network complexity in the sense that the monitoring functionality is static and predefined; therefore, there is not enough intelligence to deal with unexpected network changes.
- Static decentralised monitoring systems, such as Common Object Request Broker Architecture (CORBA) [CORBAWeb] and Java Remote Method Invocation (Java-RMI) [RMIWeb] based distributed object approaches, are decentralized in the sense of employing hierarchical infrastructure or based on distributed objects. However the monitoring functionalities are pre-defined and static. From this point of view, they have limited ability to detect network topology changes and to react to complex network behaviours. They are not scalable in the sense that pre-

installation of the monitoring system requires a significant amount of human administration which is not practical and manageable in a large scale network.

- Dynamic decentralized monitoring systems, such as Management by Delegation (MbD) [Goldszmidt95] or mobile agent based approaches, are dynamic and distributed in the sense of being programmable and decentralized with mobile code. The main idea is to exploit the code mobility to perform a measurement or an analysis of a network component without permanently consuming local resources. The scalability, flexibility and robustness are somewhat increased. However, for a large-scale, ever-changing network, the intelligence required to *locate* measurement sites and support the underlying communication infrastructure are far from being mature. More active approaches have been proposed, that attempt to maintain and adjust an agent's behaviour dynamically according to changes in the monitored systems [Liotta02]. However, to do so, such approaches must rely on routing information obtained from network routers through standard network management interfaces; this access requirement can not always be satisfied so that its applicability is constrained. Additionally, the performance of such a system depends on some "distance" metrics (delay, throughput etc.); how one obtains the values of these metrics and the quality of these metric values will affected the system's performance. Finally in practice, security concerns prevent such an approach from being widely deployed.

3.1.3 How can overlays help?

Although overlay networks aggravate the complexity and difficulty in managing current and future networks, at the same time, they appear with their advantages in addressing these problems. For example, as discussed in Section 2.3.3, some P2P networks have been successfully used in distributed file sharing applications on a global scale, and have exhibited special advantages in terms of scalability and robustness.

Examining overlay networks closely, the essential benefit they bring is that, new or enhanced network functionalities and services can be immediately deployed without impacting the underlying well formed infrastructure of IP routing in the Internet. This can be explained as follows:

- With overlay networks, designers and developers are able to develop and implement new routing and packet management algorithms quickly and easily on top of the Internet.
- Once these new services and functionalities are proven to be applicable, instead of modifying Internet protocols and having to overcome the associated technical and political hurdles, service providers can create these new services and functionalities without requiring any changes in the underlying network, nor universal adoption from network providers.
- The deployed overlay services and functionalities can be very flexible and scalable, and can quickly detect and avoid network congestion by adaptively selecting paths based on different metrics, such as the probed latency.
- Even if there are any flaws in these newly deployed services and functionalities, the underlying IP routing infrastructure of the Internet remains stable.
- Given the computing resources that current commodity hardware can provide, e.g. CPU processing capability, memory, and hard disk, expensive tasks that would normally be well beyond the ability of a conventional router can be shifted to the network edge; therefore the abundant resources at the edge of the network can be exploited to give any required hardware support.

3.1.4 At what cost?

As a trade-off, overlay networks exhibit several challenges and problems:

- Overlay networks are normally built without knowledge of the underlying physical network thus cannot have any direct control over how packets are routed in the underlying network between two overlay nodes; instead, overlay networks normally have to send probe packets to obtain underlying network information. This leads to inefficient usage of network resources, and a large number of control messages required for maintenance are generated. Additionally, due to network dynamics, such probed information still is partial and often inaccurate.
- Overlay networks are highly decentralized, thus they are unlikely to coordinate resource usage: fairness of resource sharing and collaborations among end-nodes in overlay networks are two critical issues that have not been well

addressed.

- Finally, since overlay networks are open to all Internet users, security and privacy issues can be quite serious.

Based on these observations, this dissertation argues that conventional network monitoring approaches have difficulty in addressing the following issues exhibited in NGNs: dynamic topologies, rapidly increasing scale and significant complexity. An efficient network monitoring system is therefore required to be flexible to address dynamically changing topologies, scalable to address increasing network size, and extensible to address high levels of complexity and heterogeneity; if the overhead that overlay networks normally introduce can be controlled, the advantages of overlay technologies can therefore be leveraged to build such a monitoring system. The remainder of this dissertation is focused on proving this argument.

3.2 Problem Definition

In the previous section, overlay networks are discussed in general with their pros and cons in dealing with the challenges faced by the network monitoring community. Before the discussion is extended into concrete overlay techniques and their applicability in network monitoring systems, one must analyze application requirements. In this section, OverMon, a new distributed network monitoring system built upon overlay networks, is proposed, via the definition from different perspectives, namely, deployment scale, functional requirements, performance requirements, and special features.

3.2.1 Deployment Scale

As discussed in Section 2.1.2, the Internet is comprised of a collection of Autonomous Systems (ASes). From the commercial point of view, this autonomy is essential to the current Internet's operation: the network services provided by ISPs are normally bounded between ASes, such that pricing and billing activities can be conducted easily. More importantly, from a technology point of view, such autonomy improves the Internet's scalability in that it enables engineering independence among autonomous systems. In other words, the AS-level networks are more suitable and practical to deploy valued-added services quickly and efficiently, since these services can be updated at any time without impacting other domains.

For the network measurement community, edge-to-edge measurement is defined as conducting network measurement tasks on a set of edge routers within a single administrative domain. Given the capability of the computing resources that current commodity hardware provides to edge routers, as well as the fully controlled access to edge routers, edge-to-edge network monitoring is envisioned as an efficient approach to reflect the status of network performance within a single administrative domain.

Considering these factors, OverMon is defined to be deployed on edge routers within a single administrative domain (e.g. a medium sized campus or a company's network), performing edge-to-edge measurement tasks. As a value added service deployed on edge routers, no changes to the underlying physical network are required; in particular, no modifications to core routers are required, nor are core routers aware of the existence of OverMon; of course, they are still involved in routing overlay packets at the network level.

Figure 3-1 illustrates OverMon's deployment scale from a user's view, in which $E_1, E_2 ... E_7$ are edge routers and C_1, C_2, C_3, C_4 are core routers. The measurement tasks $m_1, m_2 ... m_n$ are issued by client applications at E_1, from which the measurement parameters are distributed to all participant nodes – in this case, edge routers $E_1, E_2 ... E_6$. In this two-point probe-based measurement task, $E_1 \rightarrow E_2$, $E_4 \rightarrow E_3$, and $E_5 \rightarrow E_6$ are end-system pairs corresponding to the probe senders and probe receivers. In addition, E_3 and E_6 are the nodes where queries are issued for the results of task m_1.

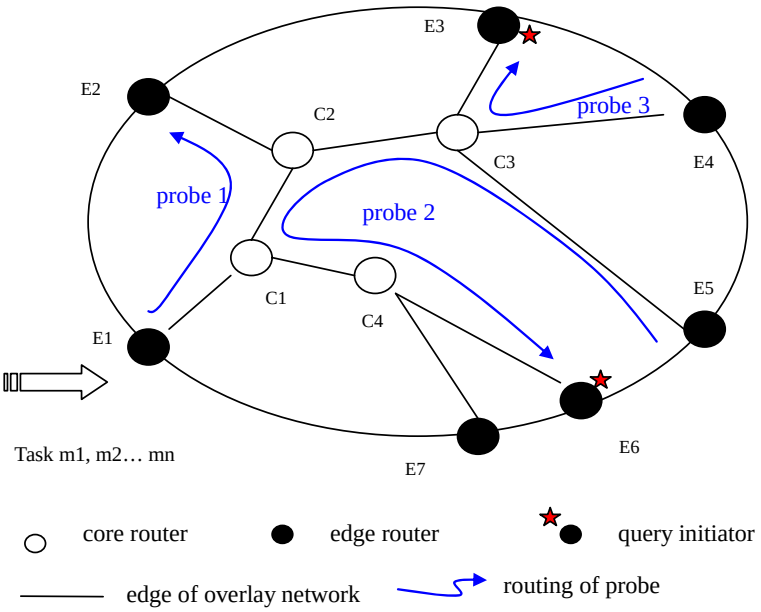


Figure 3-1: The definition of OverMon's deployment scale

3.2.2 Functionality Requirements

Traditional network monitoring approaches, such as static centralized SNMP, although having difficulty in coping with the previously mentioned new challenges, are still widely deployed due to their simplicity. Thus, from a practical view, for the initial deployment of a new paradigm of overlay based network monitoring system, it is better to *complement* existing solutions. For example, with the new overlay based network monitoring system, network administrators, or other client applications such as network management applications, can conduct a series of active measurement tasks when necessary to verify any abnormal network behaviour.

A typical usage scenario of OverMon can be that, within a single administrative domain, Cisco Netflow might be used as a passive means to provide a summary of traffic through routers. Network flows are of granular information from which analysis can be made on the application types of network traffic. If the volume of TCP/UDP traffic exceeds the normal level (e.g. by percentage of bandwidth or bits/second), a network administrator can use OverMon to measure the impact of the abnormal TCP/UDP traffic on network performance. He can select a subset of edge routers, group them into <sender, receiver> pairs, specify the metric to measure, configure the probe structure, and initiate the measurement task through OverMon to simultaneously send probe packets between the router pairs. Once the measurement task is finished, assuming packet loss rate is the metric to be measured, he can query the result by asking “From this measurement, which routers have seen packet loss rates in percent between 0.5 ~ 1.0?”. If the abnormal TCP/UDP traffic appears few times a day or on daily basis, such active measurements can be instrumented at various times or with various metric and probe structures, and more complicated queries can be issued such as “Of tasks that have been performed between 8:00 am ~ 6:00 pm, which routers have seen packet loss rates in percent between 1.0 ~ 2.0?”, or “Of tasks conducted today, which routers have seen packet loss rates in percent less than 1.5, and packet delay between 50 ~ 100 ms?”.

OverMon therefore can be envisioned as a light-weight monitoring system, which enables users to obtain timely snapshots of network performance by initiating and executing un-planned monitoring tasks upon request. After a distributed measurement task is finished, the raw measurement results are encapsulated in the form of tuples, which contain the values of performance metrics, as well as information on task configuration (e.g. task ID, measurement sites etc). These measurement results are then stored and indexed in a distributed style, which enables users to issue queries to obtain measurement results of

interest.

Figure 3-2 shows a functionality view of OverMon, in which OverMon interfaces with the client applications and the underlying intra-domain network, such that edge-to-edge active measurement tasks can be registered on demand and triggered to be performed as scheduled; the measurement results are collected and stored, and can be queried by users.

To realize these functionalities, a communication-centric overlay network needs to be constructed to propagate task configuration parameters to the network nodes that are required to participate in the measurement task; a data-centric overlay network also needs to be constructed, to store and index the accumulated monitoring results hence to support measurement results being queried.

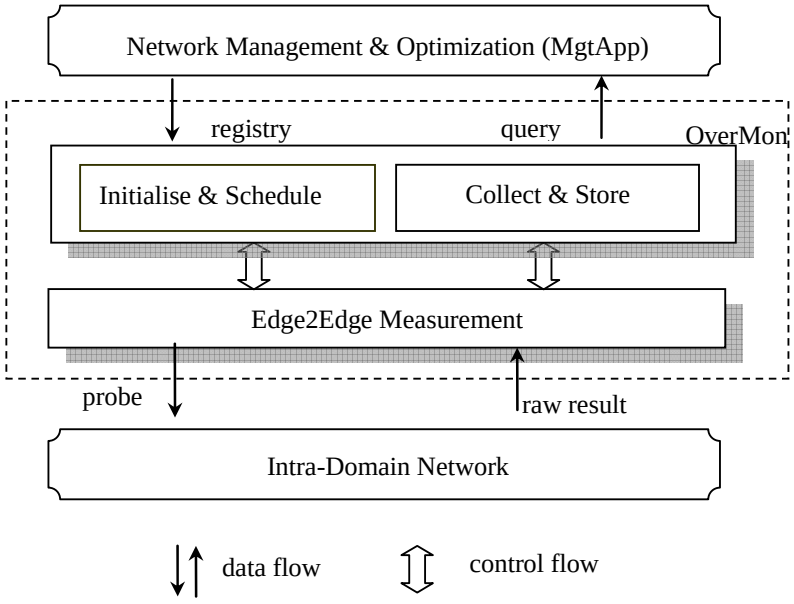


Figure 3-2: The definition of OverMon's functionalities

3.2.3 Performance Requirements

As discussed, to tackle the challenges brought by current and future networks, OverMon needs to be flexible in dealing with dynamic topology, scalable in dealing with large size networks, and extensible in dealing with complicated network behaviours.

- Out of flexibility concern, rather than being defined uniformly in advance, the monitoring tasks that a user can initiate should be dynamically configured on demand; as such, users can freely and timely initiate monitoring tasks according to

the status of network performance.

- Out of scalability concern, the distributed architecture not only should eliminate single points of failure and overload bottlenecks, but also should keep the cost of communication and computation as low as possible, and be bounded when network monitoring tasks involve a large number of measurement sites. Here, the cost of communication mainly refers to the bandwidth spent in maintaining the overlay network; while the cost of computation mainly refers to the CPU cycles spent by edge routers in overlay maintenance.
- Out of extensibility concern, the proposed architecture should be designed and implemented in a component-based approach with extensible interfaces; thus, when new functionalities are required to be added, the system infrastructure does not need to be changed.

3.2.4 Special Features

Considering the different perspectives from which OverMon has been defined, namely where to deploy, what it can do, and how good it can be, this section gathers special features derived from these definitions. With these special features, OverMon can be distinguished from other overlay systems, such that special attention should be paid in the design.

Life Time of Nodes The OverMon software is run on edge routers within a single administrative domain. Compared with end hosts of the Internet that are often used in P2P applications, the lifetime of OverMon nodes is relatively long, in the sense that edge routers are not frequently shutdown or offline, thus, the system possesses a high level of stability.

Participant nodes of Sessions Participants here is defined as all the OverMon nodes that are required to take part in a measurement task; and session here is defined as an initiation process of a monitoring task, or a query process on measurement result. Note that OverMon is not responsible for making the decision of choosing the proper network nodes for a measurement task; rather, the participant nodes are explicitly specified by users. As a result, there is no active "*join*" (or "*leave*") request put out from the OverMon nodes, thus for a single session, the participant nodes are unchanging.

Resource Advantages and Constraints Since each OverMon node is running on an edge router, not on an end host of the Internet, on the one hand, OverMon nodes are equipped

with a similar level of computation and communication resources, such that the heterogeneity issue does not exist; furthermore, special resources such as the knowledge of AS wide network topology or routing table is possible to obtain or reconstruct. On the other hand, since edge routers are often the venue from which ISPs attempt to make direct commercial profit, e.g. by running value-added services as much as possible, the resources that are available for each service are constrained, so that it requires OverMon's overhead be as low as possible.

Session Duration and Coexistence Compared with other overlay-based content delivery applications, e.g. P2P file sharing and audio/video multicasting, OverMon is *lightweight* in that the information disseminated along the overlay paths, i.e. task parameters and queried measurement results, is small in size, and the period that a session lasts is very short – it lasts as long as task parameter packets take to reach the participant nodes and the root node receives the ACK packets from participant nodes, or as long as a range query is routed until being resolved and the results are returned. However, due to OverMon's target of being completely distributed and that each OverMon node can respond to a user's request, multiple sessions might be initiated and co-exist at the same time.

Multi-Attribute Small Dataset After a distributed measurement task is finished, the raw measurement results are stored and indexed. Compared with data sets in other popular overlay applications, such as video/audio chunks for multimedia broadcasting or P2P file sharing, OverMon's dataset is smaller in size but with multiple attributes, which implies that the complexity of splitting data into chunks does not exist, however the difficulty in indexing data is increased. For example, simply hashing the whole data item might not work well, since it hides the features of the data item such that the query can only be issued against the data ID, not the data value.

3.3 Applicability of Overlay Techniques

Having defined OverMon from different perspectives, this section extends the discussion to relevant overlay techniques that might be applicable to OverMon, namely application level multicast, and overlay based data management.

3.3.1 Application Layer Multicast

This section describes application layer multicast, by discussing its emerging history, the

taxonomy of construction approaches, and the applicability of different approaches to OverMon.

3.3.1.1 Overview

Multicast is one of the network services that can be materialized on overlay networks. The problem it tackles is to distribute contents to more than one destination simultaneously, thus to save bandwidth usage.

IP Multicast is one such bandwidth-saving technology: it delivers network traffic from a source to multiple receivers without adding any additional burden on the source or the receivers, while using the least network bandwidth of any competing technology [CiscoIPM06]. Typically, IP multicast packets are transported along a distributed data delivery tree defined on the routers of the network. It uses the least network bandwidth since it is able to reduce packet replication on the wide-area network to the minimum.

However, deployment of IP Multicast has been limited due to a number of technical and non-technical reasons: 1) the setup of routing and forwarding table requires complicated configuration and maintenance; 2) an unique multicast group address is not easily obtained; 3) reliability and congestion control on top of IP Multicast is difficult to implement, thus the performance is directly impacted; and 4) the pricing model for multicast traffic is not yet well-defined.

Application Level Multicast (ALM) originally is to remedy these problems by moving the management functionalities, such as membership management, error, flow, and congestion control from the network layer to the application layer. That is, in application level multicast, multicast session members form an overlay network on top of the network layer, and multicast data are transmitted along the overlay paths.

Currently, the dominating application domain using ALM is large scale content delivery, which broadly includes audio/video broadcasting, on-line conferencing, on-line gaming etc.

3.3.1.2 Taxonomy

Typical ALM approaches involve both data plane and control plane. Here, data plane mainly refers to the mechanism of transferring data between hosts in the multicast session, while control plane mainly refers to how a multicast session is created. Accordingly, the construction of ALM can be classified by the sequence of firstly constructing the delivery tree for the data plane (i.e. *tree-first approach*), or the maintenance mesh for the control

plane (i.e. *mesh-first approach*). When the multicast tree is not explicitly built but determined by the routing algorithm of the overlay network, it is classified as an *implicit approach*. The ALM approaches built upon DHT normally fall into the implicit category; they can also be referred as *structured ALM* while the tree-first approach and the mesh-first approach can both be referred as *unstructured ALM*. In Table 3-1, the taxonomy of existing ALM solutions are summarized, followed by the discussion of representative example(s) from each category.

Table 3.1: Taxonomy of application level multicast approaches

Unstructured ALM	**Mesh First**	Narada[Chu00],OverCast[Jannotti00],HyperCast[Liebeherr99]
	Tree First	ALMI[Pendarakis01], FatNemo [Birrer04], Yoid[Francis00]
Structure ALM (i.e. DHT based)	**Flooding Based**	CAN-based[Ratnasamy01]
	Tree Based	Bayeux [Zhuang01], Scribe [Rowstron01+]

For the mesh-first approach, End System Multicast (ESM), also called Narada [Chu00], is a representative example. It firstly builds up a richly connected graph (i.e. the mesh), and tries to ensure that the mesh has some desirable performance properties; the delivery tree is then built on top of this mesh using a distance vector based DVMRP-like protocol [Waitzman88]. Typically, in a mesh-first approach, the efficiency of the resulting tree depends on the "goodness" of the mesh, in which the quality of a path between two members is comparable to the quality of the unicast path between them. To maintain the mesh, each member has a limited number of neighbours in the system, and it periodically exchanges its knowledge of group membership with its neighbours; the knowledge of group membership is obtained by probing each other at random, therefore new links may be added or dropped. Other mesh-first approaches include Overcast [Jannotti00] and HyperCast [Liebeherr99].

For the tree-first approach, ALMI (Application Level Multicast Infrastructure) [Pendarakis01] is a representative example. ALMI builds one shared minimum spanning tree (MST) between all members of the multicast group to transmit multicast data. ALMI

uses a centralized algorithm to improve the multicast tree construction: each node has a connection to a session controller node; this controller node is responsible for organising the tree structure, as well as processing members' join and leave requests. To organize the tree, the controller instructs each member to monitor a set of other members; the spanning tree is then periodically calculated based on the measurement updates received from all members. Therefore, for each session member, besides forwarding data on the data plane, it also monitors the unicast paths to, and from, a subset of other session members by periodically sending probes to these members. Other tree-first approaches include FatNemo[Birrer04], and Yoid [Francis00].

For DHT-based structured ALM [Ratnasamy01, Zhuang01, and Rowstron01], two approaches are mainly taken: flooding and tree building. The ALM built on top of CAN [Ratnasamy01] is a typical example of the flooding approach; Scribe [Rowstron01+] (i.e. based on Pastry [Rowstron01]), and Bayeux [Zhuang01] (i.e. based on Tapestry [Zhao04]) are two representative examples of the tree building approach. The flooding approach provides broadcast functionality by leveraging the information that nodes maintain for overlay routing, i.e. it constructs a separate overlay network per multicast session to propagate or broadcast messages within this overlay. Whereas the alternative tree-based approach builds a tree for each group, instead of a separate overlay network; it leverages the object location and routing properties of the underlying DHT overlay network, to create groups and join groups; the multicast messages related to a group are propagated only along its associated forwarding tree.

In summary, given that ALM is to address the lack of IP multicast deployment, it sacrifices bandwidth and processor cycles for easier deployment of multicast services without requiring global multicast-enabled IP routers. However, its performance can't be as efficient as IP multicast, such that the ultimate target of ALM is to construct an overlay network that is close to the underlying physical network in terms of topology and performance. In practice, to achieve this, one typical approach is to periodically probe the network actively to obtain updated performance information; then the control mesh and the delivery tree can be built based on these measurement results. To reduce the overheads caused by the probe traffic, some work has been done by using synthetic coordination to predict and evaluate the distance between two nodes, before making any choice of the next hop among the neighbour nodes. However, the maintenance of synthetic coordination either still relies on active probes [Dabek04, IDMapWeb], or requires landmark nodes [GNPWeb], that normally are not always available.

3.3.1.3 Applicability

Recall that OverMon needs to disseminate the configurations of a monitoring task to multiple participant nodes, Conventional point-to-point configuration approaches do not make efficient use of bandwidth, since the unicast dissemination in these conventional approaches lacks scalability in that the bandwidth spent for each destination is duplicated. By using multicast as the dissemination paradigm, task configurations can be distributed to more than one destination simultaneously, thus the bandwidth usage is efficient. Furthermore, compared to IP multicast, it is much easier for application level multicast to construct multicast sessions differently with great flexibility adapting to different monitoring tasks. Therefore, ALM is a suitable technology to apply.

Among the ALM construction approaches discussed above, mesh-first approaches [Jannotti00, Chu00, and Liebeherr99] only suit small to medium sized groups, since the overhead in maintaining the mesh with fewer nodes is not significantly high. Tree-based approaches [Pendarakis01, Birrer04, and Francis00] have the same overhead problem due to the probe traffic, especially when the size of a multicast group is large; in addition, single points of failure might exist on the tree if no protection paths are provided by the mesh or by the centralized server. Regarding DHT based solutions [Ratnasamy01,Zhuang01, and Rowstron01], the flooding approach saves bandwidth if most nodes in the existing overlay network are interested in receiving a broadcast message; otherwise for groups consisting of a small subset of the overlay network's nodes, it is not efficient to broadcast the message to the entire overlay network. In this case, either separate overlay networks have to be constructed for broadcasting, or the tree building approach should be adopted instead.

For OverMon, building ALM on DHT-based overlay can achieve good scalability due to DHT's inherent advantages of being self-organizing and self-healing; however one pays a price for the high level of overhead in maintaining the DHT. Also, due to DHT's randomness in data placement for key-matching based routing, the multicast tree has to be built with blindness of the underlying network topology. As a result, continuous and duplicate paths are likely to be taken, which will degrade the overall performance. Considering unstructured ALM approaches, the primary concern is still overhead. Given that OverMon supports on-demand style task initiation, and that it is a lightweight content distribution application, in terms of relatively small datasets, short sessions, and static membership, the mesh-first approach is not feasible due to the high overhead in maintaining the mesh, as the maintenance overhead is always incurred even there is no task

being initiated.

Therefore, the tree based unstructured ALM approach is considered; the challenge is how to build the ALM tree with good performance but low overheads.

3.3.2 Overlay Based Data Management

This section introduces overlay based data management, by discussing its emerging history, the taxonomy of different approaches, and the applicability of different approaches to OverMon.

3.3.2.1 Overview

Data management is emerging as a new network service that can be materialized on overlay networks. In the context of this dissertation, the problem it tackles is to manage data in a distributed manner by using overlay networks.

Research on this topic is a joint agenda for both the network and the database communities, since overlay networks can provide promising mechanisms for sharing and accessing large and heterogeneous collections of computing resources, however it lacks the expertise in managing the semantics of data, the transformation of data, and the relationship of data. On the other hand, databases and data management tools can provide semantic rich data representations and expressive query languages, however it lacks scalability in terms of centralized schema, and point-to-point administration mode prevents it from supporting a large number of users.

By overlapping and influencing each other, overlay-based data management is a compromise for the database community to trade relaxed semantics with good scalability shown in today's overlay networks. While for the network community, better interaction can be achieved between distributed nodes, with semantics being maintained but less dependency on a centralized schema.

3.3.2.2 Taxonomy

Overlay based data management systems can be grouped by their functionality, i.e. aggregation and query. In addition, range query is a special branch of query, that is, rather than retrieving an exactly-matched point value, range query returns all the values that satisfy the range interval as specified in the query condition. In Table 3-2, the taxonomy of existing solutions focusing on these areas is summarized, followed by a discussion of

representative example(s) from each category.

Table 3.2: Taxonomy of overlay based data management approaches

<table>
<tr><td rowspan="2">Aggregation</td><td colspan="2">Non-DHT</td><td colspan="2">Astrolabe [VanRenesse03]</td></tr>
<tr><td colspan="2">DHT-based</td><td colspan="2">SDIMS[Yalagandula04],Willow[VanRenesse04], SOMO[Zhang03],Cone[Rhagwan04],DASIS[Albrecht03]</td></tr>
<tr><td rowspan="4">Range Query</td><td rowspan="3">DHT-Based</td><td></td><td>Tree</td><td>Trie</td></tr>
<tr><td>On-top</td><td>DST[Zheng06],Squid[Schmidt03]</td><td>PHT[Chawathe05]</td></tr>
<tr><td>In-network</td><td>GIIS[Andrzejak02]</td><td>P-Grid [Anwitaman05]</td></tr>
<tr><td rowspan="2">Non-DHT Based</td><td>Skip List based</td><td colspan="2">SkipNet[Harvey03] SkipGraph[Aspnes03] SkipIndex[CZhang04]</td></tr>
<tr><td>Ring-based Hub</td><td colspan="2">Mercury[Bharambe04]</td></tr>
<tr><td>General Query</td><td colspan="4">Pier[Huebsch03], Piazza[Halevy04]</td></tr>
</table>

Aggregation

Aggregation can be abstracted as providing scalability to a large scale distributed information system by allowing a node to view detailed information about the state near it, and progressively coarser-grained summaries about larger subsets of a system's data [VanRenesse03]. Intuitively, data aggregation is simply a reversed operation of group-data distribution, that is, if group-data distribution is a process of top-down tree construction, data aggregation is a bottom-up tree convergence. In fact, the problem of data aggregation in a network system involves both the data plane and the control plane: from the data plane, overlay based aggregation determines which and how the attribute values are aggregated, i.e. the merging of data using statistical and (or) mathematical processing functions; from the control plane, the systematic level design determines how to store, identify and transmit

the data according to the aggregation hierarchy that is determined by the aggregation functions. Note that the design of the data plane for aggregation is quite application specific. For instance, for a network performance monitoring system, the target attributes to be aggregated are network performance metrics, such as throughput, packet delay, packet loss, and jitter; the aggregation functions can be simple calculations, such as min, max, average, and count.

Significant research has focussed on the design of an aggregation abstraction. Astrolabe [VanRenesse03] provides the abstraction of a single logical aggregation tree that mirrors the system administrative hierarchy. Astrolabe is robust in the sense of using an unstructured gossip protocol for information dissemination, along with its strategy of replicating all aggregated attribute values in the sub-trees. However, the high degree of replication may limit the system's ability to accommodate large numbers of attributes. Furthermore, the essential feature of gossip protocols introduces substantial end-to-end latency, as at a particular instant of time, the set of states that a node holds, might be stale, thus not accurate and reliable.

Other P2P based aggregation systems have been proposed, including SDIM [Yalagandula04], Willow [VanRenesse04], SOMO [Zhang03], Cone [Rhagwan04], and DASIS [Albrecht03]. They are all built using DHT based protocols; two different tree construction approaches have been used: either a single tree is built for all the attributes [VanRenesse04, Zhang03, and Albrecht03], or separate trees are built for each attribute [Yalagandula04, Rhagwan04]. Essentially, DHT-based aggregation approaches follow Plaxton's routing infrastructure and augment the prefix-matching mechanism to build aggregation tree [Rowstron01]. In these systems, each key identifies a tree consisting of the route from each node to the root node for that key; the values of an attribute at a parent node are naturally determined by the associated aggregation functions over the values at its child nodes. The challenge lies in putting new nodes at the right position, processing the merger of disjoint trees, and developing flexible aggregation functions that are supported by tree walking.

General Query

In parallel with distributed aggregation as discussed above, distributed query has also attracted considerable attention. Efforts have been put into preserving rich semantics via a distributed data management architecture, in which any user can contribute new data, schema information, or even mappings between other peers' schemas over an unstructured overlay network [Halevy04].

Towards this direction, PIER (Peer-to-Peer Information Exchange and Retrieval) is designed to explore the application of general semantics to distributed P2P systems [Huebsch03] by using CAN, the DHT-based overlay network [Ratnasamy01]. In that work, PIER is a three-layered distributed query engine: the bottom DHT layer is responsible for scalability of dynamic content routing; a storage manager at the middle layer is responsible for temporary storage management of data that are contributed by the nodes connected to the system; and the top application layer is responsible for integrating the routing layer and storage manager layer, and for providing an interface to application developers.

Range Query

Beyond general query, range query is important to many distributed applications, especially those with discovery and exploration purposes, such as urban traffic monitoring or network performance monitoring. Typical range queries in a network monitoring system can be “retrieve all of the host names where the packet loss rate is larger than 0.5” or “retrieve all of the host names where the average packet delay is between 30 ~ 60 ms”.

Given the overlay networks that distributed range query can be built upon, naïve approaches are to set up a central index or to flood the whole system. Both are easy to build, however they all suffer scalability issues due to unbounded overhead.

To improve such unstructured solutions, DHT is one of the popular platforms that can be leveraged to build distributed range query: either on-top of DHT, or in-network of DHT. The on-top approach refers to constructing a separate data structure purely for range query purposes; the data structure is infused with the underlying distributed hashing table by using DHT’s lookup interface purely for routing. Alternatively, the in-network approach refers to making the DHT itself as a data structure supporting range query; in other words, to *modify* DHT by moving DHT peers, and adaptively change peers’ identifiers to dynamically match the changing distribution of the data. For example, DST (Distributed Segment Tree) [Zheng06] is built on top of a DHT using the segment tree data structure to maintain ordered ranges; the range interval [*s*,*t*] is assigned to a DHT peer that is associated with the key *Hash*([*s*,*t*]). Similarly, PHT (Prefix Hashing Tree) [Chawathe05] is another approach built on-top of DHT, but with a different data structure: a PHT is a normal binary trie that integrates with the underlying DHT by hashing the prefix labels of a PHT node over the DHT identifier space. Regarding in-network approaches, P-Grid [Anwitaman05] and GIIS (Grid Index Information Service) [Andrzejak02] are two examples. P-Grid makes the overlay network itself a trie; the hashing function in P-Grid is order-preserving in that data items that are semantically close remain close to each other and the ordering

relationship of keys (i.e. the prefix of keys) are preserved after being hashed. GIIS (Grid Index Information Service) is an in-network approach extended from CAN, a DHT in the form of a d-dimensional Cartesian Space; each node in GIIS acts as an Interval-Keeper (IK) and shares the responsibility of maintaining a certain sub-interval of the whole range of the attribute values, as well as the zone in the logical d-dimensional space.

Non-DHT based solutions have also been investigated, for example, the skip list based approach. A skip list is a probabilistic data structure consisting of a collection of ordered linked lists by which data can be placed according to any predefined order. These ordered lists form into different layers and all nodes participate in the bottom layer 0 list; recursively, starting from the layer 0 list, with some probability, a subset of lower layer nodes participate in the upper layer list. The process of looking up data starts from the higher sparse layer and comes down to the lower dense layer. Examples of constructing range queries using the skip list data structure include Skip Graph [Aspnes03], SkipNet [Harvey03], and SkipIndex[CZhang04].

Another non-DHT based data structure for distributed range query is the ring based hub, as proposed in Mercury [Bharambe04]. In Mercury, to handle multi-attribute query, a ring based routing hub, i.e. a logical collection of nodes in the system, is created for each attribute; multiple hubs can be thought of as orthogonal dimensions of multi-dimensional attribute space. Within each hub, to handle range queries, each hub is organized into a circular overlay network with data being placed contiguously on the ring; each node is responsible for a continuous range of values for the particular attribute.

Note that Mercury is not the only solution dealing with multiple-attribute range query. Among DHT-based solutions such as GIIS [Andrzejak02], Squid [Schmidt03] and PHT [Chawathe05], *Space Filling Curve* (SFC) is used to tackle this issue. SFC is a linearization technique typically used for dimension reduction, image processing, and sparse multi-dimensional database indexing. Essentially, a SFC is a special case of fractals [Lawder00]; it divides the space into a number of line segments, visits the line segments in a specific order, and maps a unit line segment to a continuous curve in the unit square, cube, and hypercube etc [Lawder00]. By dealing with one fragment at a time, SFC possesses the property of locality preservation, such that for a multi-attribute range query, SFC is normally used to map multi-dimensional data into a single dimensioned key space, e.g. the ring based DHT Chord; and then network nodes are arranged linearly along the ring.

Finally, load balancing is important in keeping a distributed system resilient to the failures of a small number of overwhelmed nodes. As a new application domain, P2P networks

have raised considerable interest in this topic [Godfrey04, Aspnes03, and Bharambe04]. For structured overlay based range query, i.e. both DHT based and non-DHT based approaches, but not flooding and centralized approaches, the loading balancing issue always needs to be addressed, due to the loss of data locality[5] caused by random placement.

3.3.2.3 Applicability

As seen from Table 3-2, several overlay based data management solutions have been proposed, which seems to provide sufficient candidates from which OverMon can choose. Given that OverMon is defined as a light-weight network monitoring system allowing users to execute on-demand active measurement tasks, it requires the flexibility of obtaining measurement results in a timely and agile fashion. Thus, the primary criterion for choosing an applicable solution comes down to the consideration of flexibility.

For overlay-based aggregation, most proposed solutions only support the formation of a continuous aggregation hierarchy, which might not suit active measurement tasks, since the results generated from different active probing tasks are configured with different parameters for different purposes; simply applying a continuous aggregation function on multiple task results might hide the result features desired by each task. Even if it is meaningful to aggregate raw results across multiple tasks, or from multiple sites for a particular monitoring task, since users are given the full control of configuring active probes, the aggregation function needs to be highly variable for different measurement purposes. However, this on-demand style aggregation is not supported by the proposed aggregation solutions. Furthermore, out of overhead concern, this continuous aggregation is not economical in terms of utilizing network resources, since the aggregation hierarchy has to be maintained even there is no any active probing task initiated and no results produced.

In contrast, distributed query, particularly distributed range query, seems to provide this flexibility: it allows users to obtain the snapshot of network performance on the fly - by simply specifying the changeable and adaptable query conditions. Additionally, it facilitates the extensibility of OverMon: when OverMon is extended to support passive measurement or other measurement techniques, this advantage of flexibility still holds. Certainly it is interesting to explore how continuous aggregation and on-demand range query can co-exist in OverMon; however, the discussion of this topic is out of the scope of

[5] Locality here refers to the relationship between data, e.g. closeness in value, or ordering in sequence.

this dissertation.

Examining existing range query solutions, with respect to the DHT-based solutions, although a system can inherit the advantages of scalability and robustness from DHT, the overhead that is introduced is very large. This is due to the fact that intuitively to support range query, it is better to *firstly place the data in an orderly fashion*. DHT, on the contrary, *places data blindly* without considering the relationship of data, or the underlying network topology. Thus, the scalability and robustness is achieved through losing the locality of data. Therefore, to efficiently support range query, extra indexing effort has to be expended to map the data space to the DHT identifier space, which unavoidably introduces considerable overhead. In addition, when DHT based solutions deal with multiple attribute range query using space filling curve, a locality preserving curve is not trivial to define, and the complexity increases exponentially with the number of dimension [Lawder00]. Therefore DHT based solutions are not suitable for OverMon.

Examining the rest of the non-DHT based candidates, given that the features of datasets in OverMon are multi-attribute, hub based Mercury seems to suit best, as Mercury supports multi-attribute range query; and the hub-based data structure is simpler and easier than the skip-list data structure to construct and maintain, which can benefit the edge routers in saving memory/CPU usage.

3.4 Related Work

As discussed in Section 3.1, distributed architectures have been widely accepted as the direction for future network monitoring systems. Recently, overlay networks, typically represented as P2P networks, have attracted tremendous attention from the network research community. What follows is a description of a few examples of the application of overlay technologies to the network management domain.

Authors in [Habib04] design a SLA validation monitoring system, in which all edge routers form an overlay network and probe packets are sent along the overlay paths to identify all the congested links with high loss; by using tomography, the clients who invade and exploit extra usage than specified in the SLA can be inferred. The focus of [Habib04] is different from this work since in OverMon, the probe packets are actually sent along unicast paths; the overlay paths are used to transport the configuration parameters to the participant sites, and to route measurement results/queries for data management. .

The work in [Prieto06] is to build up a distributed network monitoring paradigm to

aggregate and transport measurement results on a spanning tree overlay; to avoid the bottleneck problem around the root node area in the spanning tree, filters at each node are dynamically configured according to a stochastic model, to drop unnecessary updates and aggregates. Their work is focused on continuous monitoring to support adaptive network management. That is, objects are continuously monitored in a passive way but only meaningful updates of objects are aggregated. To do so, the query semantics have to be pre-defined. In addition, it assumes that the underlying services, such as failure detection service and neighbour discovery service, exit and are reliable. Different from this work, OverMon's focus is not using a stochastic model to infer the updates of performance; instead, OverMon gives full control on obtaining result to users, such that a query can be defined to meet the user's management requirement at run-time with significant flexibility.

In [Padmanabhan05], NetProfiler was motivated by the desire to use the resources at the end of wide-area networks (e.g., the Internet), to perform network monitoring in a passive and silent style. It enables end users to share network performance information over a P2P network, with the passively observed information being aggregated along a set of logical aggregation hierarchies; and these hierarchies are predefined and associated with end hosts and their DHT-based network connectivity. The focus of that work is to set up this passive monitoring paradigm and classify different performance metrics with different aggregation characteristics, such that a suitable analysis approach can be chosen and the results can be fed back to users. OverMon is different in that it is centred on providing users the flexibility – users are not associated with any pre-defined aggregation function; rather, they can perform measurement tasks in an on-demand style, to initiate new measurement tasks and to retrieve measurement results.

In [Binzenhöfer06], the authors proposed a P2P-based framework for distribute network management. In it, the main software module called DNA (Distributed Network Agent) is running as a daemon responsible for the communication between users and individual test modules, through which local and distributed tests can be conducted. The underlying P2P layer is DHT based, using the algorithm of Kademlia [Maymounkov02]. The differences between DNA and OverMon include: DNA is aimed at end-to-end deployment scale, and the focus is to leverage the DHT layer to enable fast searching of random and specific communication partners; while OverMon is aimed at edge-to-edge deployment scale; the task participants are specified in the configuration stage, and measurement results can be retrieved by range query.

Other relevant systems include PlanetSeer [MZhang04], a diagnostic tool to detect, isolate, and classify anomalous network behaviour in a large scale environment. PlanetSeer

combines passive monitoring with active probing to form a closed loop for anomaly identification; furthermore, RON [David01] is used to monitor the functioning and quality of the Internet paths between the RON nodes, and use this information to decide whether to route packets directly over the Internet or by way of other RON nodes, to optimize application-specific routing metrics. OverMon has different foci from PlanetSeer: firstly, rather than anomaly diagnosis of routing paths, OverMon monitors the edge-to-edge network performance; secondly, OverMon is not targeting at forming a closed management loop, rather, the configuration and the analysis are left open to users.

3.5 Summary

This chapter has critically analysed the problem that is to be solved.

It firstly discusses why an overlay based network monitoring system is needed. Briefly, current and future networks bring new challenges to the network monitoring community; although overlay technologies have shown their advantages in addressing these challenges in other application domains, normal overlay networks are likely to introduce huge messaging overhead; by controlling these overheads, i.e. by making overlays bandwidth-efficient, an overlay based monitoring system should therefore be able to address the new challenges.

With this hypothesis, a new network monitoring system, OverMon, is then defined from different perspectives, such as deployment scale, functionality requirements, performance requirements, and special system level features. In short, OverMon is an edge-to-edge network performance monitoring system built using overlay networks; it supports active measurement tasks being initiated at any node in the system, and also supports corresponding measurement results being retrieved through range query.

Then in Section 3.3 and Section 3.4, relevant overlay technologies and related research work have been discussed respectively. In Section 3.3, two overlay technologies that potentially can be applied by OverMon are discussed: communication-centric overlay for control plane and data-centric overlay for data plane; both are discussed with respect to their high level introductions, taxonomies on existing solutions, and their applicability to OverMon. In Section 3.4, the existing network monitoring systems that use overlay technologies in some ways are discussed, with respect to their relevance and difference to OverMon.

The problem that OverMon targets to solve is clear: it is to investigate the feasibility of

building a network monitoring system using overlay networks; the resulting system should be flexible to address dynamically changing topologies, scalable to address increasing network size, and extensible to address high levels of complexity and heterogeneity; at the same time, by taking the advantage of the special resources at edge routers, the resulting system should not cause a high level of maintenance overheads.

Chapter 4

Design

This chapter presents the design details of OverMon. It begins with an overview in Section 4.1, introducing OverMon's design principles and system architecture. Then, Section 4.2 presents the network topology tracker. It passively snoops OSPF packets and reconstructs topology information for the constructions of control overlay and data overlay, which are discussed in Section 4.3 and Section 4.4 respectively. Section 4.5 discusses possible design alternatives. Finally, Section 4.6 summarises the contents of this chapter.

4.1 Overview

In this section, the design principles of OverMon are firstly introduced, then the system architecture of OverMon is presented.

4.1.1 Design Principles

Based on the previous analysis, i.e. the problem that is to be solved, and the applicability of overlay techniques, OverMon's design principles can be stated as follows:

1. To apply existing overlay techniques to meet the functionality requirements of OverMon,

2. To minimize the bandwidth overhead caused by the introduction of overlay networks, to meet the performance requirements of OverMon,.

These principles then guide the detailed design that will be introduced in the rest of this chapter, as well as the prototype implementation and the evaluation approaches that are introduced in the following chapters.

4.1.2 System Architecture

The system architecture of OverMon is depicted in Figure 4-1. OverMon provides functionalities that enable the clients of OverMon to register new measurement tasks, and to retrieve measurement results. The registration of new tasks can be initiated at any edge router in the system. Measurement result retrieval takes two forms: pull-based range query with query condition specified, or push-based report when a registered measurement task finishes. By using a pull-based approach, a query can be issued from any node within the system, while by using a push-based approach the reporting point is normally the node at which the task is registered. To release a user's thread from waiting, the API is RMI-based, to use an asynchronous call-back style to send users the results of registering a new measurement task or retrieving a measurement result.

Inside OverMon, the essential feature is that two overlay networks are constructed, namely the *control overlay* and *data overlay*, corresponding to the communication-centric overlay and data-centric overlay respectively, as discussed in Chapter 3, and shown in the right-hand side of Figure 4-1. Here, *control overlay* refers to an application level multicast tree along which measurement task parameters are disseminated from the task initiator to the task participant nodes; while *data overlay* refers to the overlay network by which measurement results are stored and indexed, according to data's attributes and data's locality, so as to support distributed multiple-attribute range query.

To do so, the coordinator layer on the top governs the coordination between the different functional modules. For example, *Measurement, Register*, and *Query* are the three major modules, which respectively are in charge of conducting active measurement, initiating a new measurement task[6], and organizing range queries on measurement results. They all interact with the *Virtual Repository*, logically the storage provided by the data overlay which physically consists of the edge routers' memory. That is, once task configuration parameters reach a participant node, and the new task is successfully registered with the task participant node, the task will be executed according to the configured time schedule; once the measurement task is conducted and completed, the measurement results with associated task information are then stored in the virtual repository, in the form of data records, which are tuples with multiple attributes, e.g. (*task_ID, execution_time, task_results*); range query against one or more attributes can then be conducted on these

[6] For a task initiator node, it is also responsible for *pushing* the measurement results back to user once the task finishes.

data records.

To efficiently build overlay networks, the module called *Topo.Tracker* on the left-hand side in Figure 4-1 is designed. By passively snooping the OSPF packets processed by the edge routers, followed by parsing and reusing the topology information encapsulated in the OSPF packets, overlay network construction in OverMon is more efficient since the underlying physical topology can be used, such that circuitous and duplicate overlay paths can be reduced or even avoided. More importantly, at the same time of minimizing the impact caused by the overlay networks to the underlying physical network, this performance improvement is achieved for free - it does not cause any overhead traffic; this contrasts with sending detecting-probes, an approach that is normally adopted to facilitate overlay construction and maintenance.

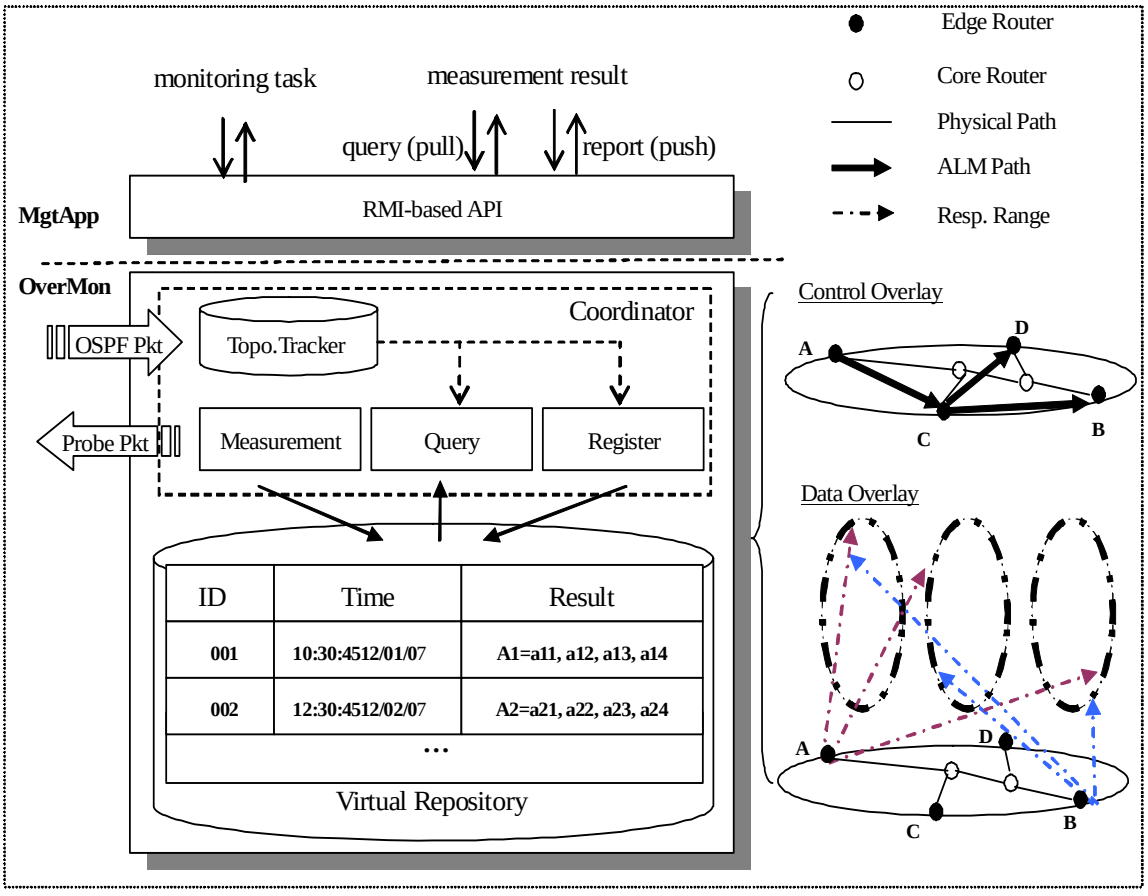


Figure 4-1: OverMon's system architecture

4.2 Network Topology Tracker

This section describes the feasibility study of snooping and parsing OSPF packets in re-constructing a network topology that is catered for OverMon overlay construction. It firstly introduces the basics of the OSPF routing protocol, then presents the concrete methodologies in reconstructing the targeted topology graph.

4.2.1 OSPF Routing Protocol

In this section, a general overview of the OSPF routing protocol is introduced, followed by the description of OSPF's topology maintenance and routing mechanisms.

4.2.1.1 OSPF Overview

As mentioned in Section 2.1.3, as a member of the family of link-state routing protocols, OSPF is an interior gateway protocol running within a single AS for IP networks. The OSPF routing table is calculated by using the Dijkstra algorithm to construct a shortest-path tree based on the type of service (TOS) metric associated with each link. The weighted link-state information is encapsulated in Link State Advertisement (LSA) packets, which collectively form the link-state database in a router's memory, and describe the network topology.

Conceptually, an OSPF topology can be viewed as a directed graph. The vertices of the graph consist of routers and networks. That is, a graph edge connecting two routers indicates that they are attached via a physical point-to-point link; and an edge connecting a router to a network segment indicates that the router has an interface on the network segment. Broadly speaking, a network segment can either be *transit* or *stub*: a transit network is represented by a vertex having both incoming and outgoing edges, and is capable of carrying traffic that is neither locally originated, nor locally destined; whereas a stub network is a vertex only having incoming edges that are locally destined.

In a network where routers run the OSPF protocol, routers firstly discover each other to establish the neighbour relationships; then *adjacency* relationships can be further formed between neighbouring routers, such that they can exchange routing information with each other to build an entire view of the network topology. Here, neighbouring routers can be seen as two routers that have interfaces to a common network segment; the neighbouring relationship is dynamically discovered and maintained by the OSPF Hello protocol,

depending upon the type of the network and the number of routers having an interface to the network. Note that the adjacency relationship is not equal to the neighbour relationship since not every two neighbour routers become adjacent - only a subset of neighbours relationship by which routing information are exchanged for an adjacency relationship.

To address the issue of scalability, the concept of *area* is introduced into OSPF, by which a set of network segments are grouped together to stand for an IP sub-netted network. For the rest of an AS, the topology of an area is hidden, and the routing within the area is determined only by the area's own topology. Since OSPF floods the network hop by hop, i.e. as soon as there are network changes, the changes are sent to every router, and then every router notifies every other router about the changes, this hiding isolates flooding traffic within an area, thus the overall routing traffic is significantly reduced. Normally, a two-layered hierarchy is formed by the backbone area (i.e. area 0), which resides at the top level, and the non-backbone areas, which reside at the bottom layer. Non-backbone areas are connected by the backbone area.

When an AS is split into areas, according to their roles in OSPF routing, the routers are divided into the following four overlapping categories:

- Internal Routers (IR): a router with all directly connected network segments belonging to the same area. These routers run a single copy of the basic routing algorithm.
- Area Border Routers (ABR): a router that is attached to multiple areas running multiple copies of the basic algorithm, with one copy for each attached area. It also condenses the topological information of their attached areas and distributes them to the backbone area; the backbone area continues to distribute this topology information to other areas.
- Backbone Routers (BR): a router that has an interface to the backbone area. It includes all routers that interface to more than one area (i.e., Area Border Routers). However, backbone routers do not have to be area border routers.
- AS Boundary Routers (ASBR): a router that exchanges routing information with routers belonging to other ASes, and it advertises external AS’s routing information throughout the local AS. It can be an internal or a area border router, and may or may not participate in the backbone area.

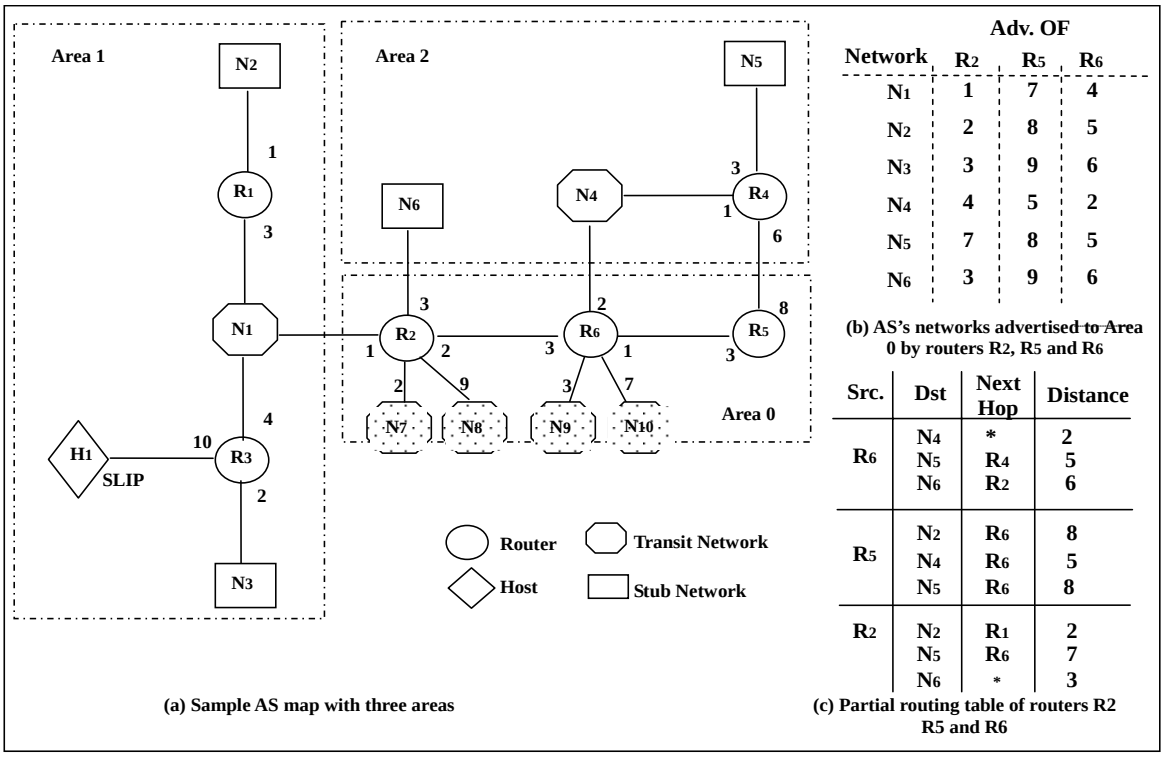


Figure 4-2: A sample AS map with three areas configured

Figure 4-2 (a) illustrates an AS map which is separated into three areas (numbered 0, 1 and 2), with six networks (numbered 1, 2…6) and one SLIP[7] link connected. Routers R1 and R3 are IR since their network connections are all in area 1. Routers R2, R4, R5 and R6 are ABR since each of them has connections to both area 0 and area 1. At the same time, R2 is a backbone router since it connects area 1 and area 2, two non-backbone areas. Lastly, routers R2 and R6 are configured as ASBR, responsible for distributing the routes received from other ASes throughout its own AS, as shown in Figure 4-2, the routes to networks N7-N10 belonging to other ASes.

The topology graph in OSPF is weighted with each edge having a cost value assigned at the configuration stage (i.e. each interface of an OSPF router is weighted). By using this weighted topology graph, each router calculates a shortest path tree with itself as the root spanning all other nodes in the graph, thus to calculate the *next-hop* for all possible destinations. In Figure 4-2 (a), a weight is depicted as a numeric label at the source of an edge, indicating the cost to the other end of the edge from the interface that the router is

[7] SLIP is a way of connecting a host to the Internet via phone lines. Without otherwise stated, SLIP is treated as a stub network.

configured; when there is no value labelled, it means the cost is zero.

Note in Figure 4-2 (a), the graph is directed with one edge actually standing for two edges with one in each direction. Taking the edge between R6 and R5 as an example, the two labels on each end of the edge actually stand for { $R_6 \xrightarrow{1} R_5, R_5 \xrightarrow{3} R_6$ }; while the edge between N4 and R6 actually stands for { $R_6 \xrightarrow{2} N_4, N_4 \xrightarrow{0} R_6$ }.

To run the OSPF protocol properly, five types of packets are defined and exchanged by OSPF routers, as listed in Table 4-1.

Table 4.1: Type, name, and functionality of OSPF packets

Type	NAME	FUNTIONALITY
1	Hello	For discovering/maintaining neighbours, in order to periodically check status of adjacent routers.
2	Database Description	For summarizing database contents, in order to describe the link state database for synchronization purposes during adjacency formation.
3	Link State Request	For downloading the database, in order to request missing or stale pieces of the link state database during adjacency formation.
4	Link State Update	For updating the database in order to send one or more LSAs to adjacent routers as a part of the flooding protocol.
5	Link State ACK	For flooding acknowledgment of receiving LSU.

4.2.1.2 OSPF Topology Maintenance

The directed and weighted topology graph in OSPF is represented in the form of a link-state database (LS database). An individual record in the LS database describes a router's local state (i.e. one of router's usable interfaces and correspondingly the reachable neighbour), as well as the type of the network that this interface connects to.

Regarding a router's local state, OSPF uses different types of LSA for describing different parts of the topology: each router describes the links to all the neighbouring routers in a given area in a ***router LSA***; an ABR originates a separate router LSA for every area that it is connected to, and summarizes the information about one area into another by originating ***summary LSAs***; an ASBR originates ***external LSA***s to describe external routing information; and a ***network LSA*** is used to describe routers attached to a network segment.

Regarding the network types that LS database records describe, OSPF supports three types of physical networks: point-to-point, broadcast, and non-broadcast. A point-to-point

network joins a single pair of routers (similarly, a SLIP directly connects a router with a host, as depicted in Figure 4-2 (a)). Both broadcast networks and non-broadcast networks allow two or more routers to be attached; the difference is how these routers discover and maintain neighbouring relationships between them. For broadcast networks, one Hello packet can be sent to all attached routers simultaneously, while non-broadcast networks require static configuration of neighbouring routers to which a router can send Hello packets to. Non-broadcast networks can further be divided into two categories, namely non-broadcast multi-access (NBMA), and Point-to-MultiPoint. The former simulates the operation of OSPF on a broadcast network, while the later treats the non-broadcast network as a collection of point-to-point links.

To achieve an identical topology view, the link-state database is synchronized throughout the AS by flooding the Link State Update (LSU) packets. OSPF routers originate and flood appropriate LSAs when network topology changes, with each LSU packet containing one or more LSAs. Note that external LSAs originated from ASBRs are flooded in the entire domain irrespective of area boundaries, hence having a domain-level flooding scope; while all other LSAs are flooded at area level.

Apart from flooding LSUs when there are topology changes, OSPF also employs periodic refreshment of LSAs when there is no any change; that is, each router periodically floods self-originated LSAs with a default refresh-period of 30 minutes. Note that the interval of noticing a routing change is normally determined by the frequency of neighbours' exchanging of *Hello* packets, as following a Hello packet, the adjacency might be reformed and the link-state database might be refreshed by exchanging new LSU packets.

To make the flooding reliable, OSPF routers reply to Link State Acknowledgment (LSA) packets directly. In other words, the flooding is performed via hop-by-hop unicast - by sending LSU packets directly to the other end of the adjacency, rather than being routed.

4.2.1.3 OSPF Routing Mechanism

An OSPF router performs routing by checking the routing table, which is the result of running Dijkstra calculation - by which, the router calculates the shortest path to any destination and the next hop to each destination is stored in the routing table.

In general, when an OSPF router has interfaces connecting to different areas, a separate copy of the OSPF protocol is run for each area (RFC 2328). In other words, for ABRs that belonging to multiple areas, they maintain multiple copies of the topological databases – one for each area that they connect to. Two OSPF routers belonging to the same area have,

for that area, an identical area link-state database. By default, multiple areas are connected by area 0; therefore typically the databases that an ABR maintains include the one for the backbone area.

In an OSPF network that has no OSPF areas configured, the process of calculating SPT within the AS is straightforward, since each router in the AS has an identical link-state database, hence leading to an identical topology graph. When multiple areas are configured, each router maintains a different link-state database, and the SPT calculation is more complicated. In that case, routing takes place at two levels, depending on whether the source and the destination of a packet reside in the same area (i.e. intra-area routing) or different areas (i.e. inter-area routing).

In the case of intra-area routing, packets are routed solely based on the information that is obtained within the area; no routing information that is obtained from outside of the area can be used. On the other hand, in the case of inter-area routing, it is the area border routers' (ABRs) responsibility to advertise to non-backbone areas the distances from ABRs to all destinations external to the area. That is, for remote areas, (i.e. the areas in which the router does not have links), an internal router (IR) does not learn the entire topology of that area, but instead, the total weight of the shortest paths from one or more ABRs to each node in remote areas. Thus, after computing the SPT tree for each area, the IR learns which ABR to use as an intermediate node for reaching each remote node.

Additionally, ABRs advertise the location of the AS boundary routers (ASBRs), thus the AS-external-LSAs advertised by ASBRs from other ASes can be flooded throughout the local AS. These LSAs are included in non-backbone areas' databases, which enables IRs to determine the shortest path to networks that are in other ASes. In return, ABRs also summarize non-backbone areas' topologies to the backbone area; such information is then advertised by ASBRs to external ASes.

For example, in Figure 4-2 (a), all the routers know that R_2 has two external routes to the network segments N_7 and N_8, with cost 2 and 9 respectively; similarly R_6 has two external routes to the network segments N_9 and N_{10}, with cost 3 and 7 respectively; router R_1 can decide between router R_2 and R_6, for routes to the network segment N_7-N_{10}. Figure 4-2 (b) illustrates the summaries advertised by ABRs R_2, R_5 and R_6, for the network segments contained in Area 1 and Area 2 (i.e., N_1-N_6). Figure 4-2 (c) illustrates the resulting partial routing table. For example, the cost from R_6 to N_5 is advertised as 5 - as the result of adding the weight value 2 of the interface on router R_6 connecting to network N_4, with the weight value 3 of the interface on router R_4 connecting to network N_5. Correspondingly,

R_6's routing table shows that the next hop to network segment N_5 is router R_4 with cost 5.

Note that OSPF uses different approaches dealing with different types of external metric values, yet their discussion is out of the scope of this dissertation. More detailed information can be obtained from (RFC 2328).

4.2.2 Topology Graph Construction

After the high level introduction to the OSPF protocol, this section discusses how the topology graph is constructed in OverMon by snooping OSPF packets. The whole task can be de-composed into five sub-tasks, namely defining the vertices and edges of the target topology graph in OverMon, choosing a proper tracking method, identifying the relevant OSPF packets, parsing the information contained in OSPF packets, and maintaining the constructed topology graph. Each of them is discussed as follows.

4.2.2.1 Topology Graph in OverMon

Given that the topology tracker provides a topology graph to OverMon for overlay constructions, this section answers the question of what the vertices and edges stand for in this topology graph.

As mentioned, the vertices in an OSPF topology graph consist of routers and network segments. Here, *network* is stressed as that is the target for routers to forward an IP packet from its source to the destination. However, OverMon is different in that it is to be deployed on edge routers only - intuitively, it implies that *networks* are not necessarily to be included in the resulting topology graph.

Actually, as noted in Section 4.2.1.1, the types of networks in an OSPF topology can largely be divided into two groups, namely stub networks and transit networks. For a stub network, it is the end of a routing path; while for a transit network, although it has both incoming and outgoing edges, and acts as a transmitting point of a routing path between routers, its outgoing edges are always weighted as zero cost. For instance, the partial topology of the AS map shown in Figure 4-2 is illustrated in Figure 4-3, only containing sample transit/stub networks with corresponding router/network LSAs. In sub-graph (a), network N_1 is a transit network with routers R_1, R_2 and R_3 attached, while network N3 is a sub network only having router R_3 connected. In sub-graph (b), network N_1 and N_3's network-LSAs as in link-state database are illustrated: in the column of N_1, there are three entries from N_1 to R_1, to R_2, and to R_3 respectively, with each weighted as zero; while in

the column of N_3, since it is a stub network, although R_3 is connected to it, there is no entry from N_3 to R_1. In sub-graph (c), router R_3's router-LSAs are illustrated, in which router R_3 has a router-LSA describing routes to N_1, N_3, and H_1 respectively, each with a non-zero weight.

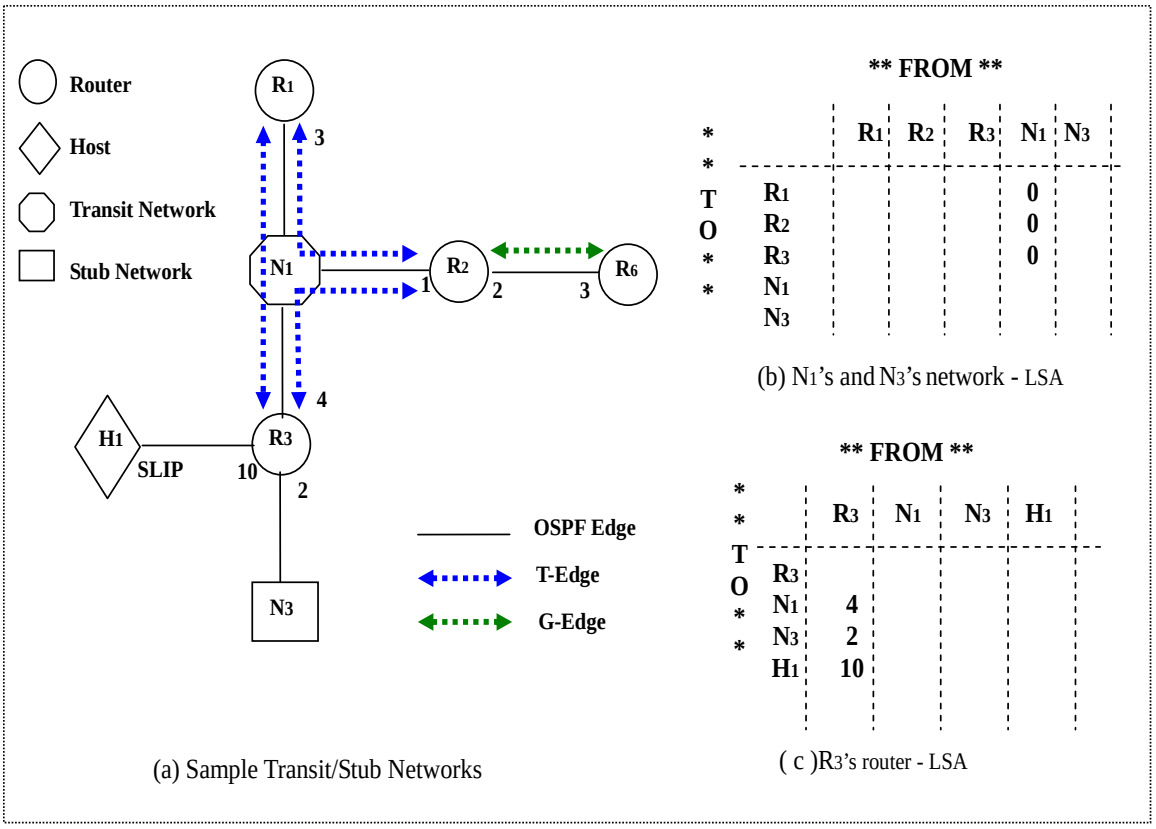


Figure 4-3: The sample transit/stub networks with corresponding router/network LSAs

Therefore, an OSPF topology graph including transit/stub network as vertices can be transformed into a topology graph containing router vertices only. To do so, a stub network can be excluded without causing any impact on the connection of routers. For a transit network, it can be considered as a transit point, such that it can be skipped over by stretching each incoming edge of a transit network to the source end of all the other incoming edges, with the weight value remains unchanged. By doing so, any connecting relationship between two routers will not get lost or changed; such a transformation reflects the physical reachability between routers, and an edge in the the resulting topology graph describes the flow of IP packets being routed between routers.

As a summary, the resulting transformed topology graph can be described as follows:

- **Vertex** The vertices contained in the OverMon topology graph only consist of

routers. The naming of each vertex comes from OSPF protocol; that is, each router is named after *Router ID (RID)*, a 32-bit numeric value that uniquely identifies a router in the AS. Normally, the largest or smallest IP address assigned to the router is chosen as it's RID. For example, if a router has two interfaces assigned with IP address *192.168.1.2* and *192.168.2.4* respectively, the later will be the RID if the algorithm is to choose the largest one. Note that the RID is only calculated at boot time or anytime the OSPF daemon is restarted.

- **Edges** An edge in the OverMon topology graph stands for the *single hop* connection between two routers: either it maps a genuine direct connection between two routers, or it is formed though the skipping of a transit network that connects multiple routers. In both cases, it remains to be directed and weighted as it does in OSPF topology graph. For simplicity, the former type is termed *G_Edge* (i.e. graph-edge), while the later is termed *T_Edge* (i.e. transformed-edge).

In Figure 4-3, the edge between router R_2 and R_6 is an example of a G_Edge. In the case of T-Edge, taking network N_1 as an example, N_1 has three incoming edges which can be depicted as $\{ R_1 \xrightarrow{3} N_1, R_2 \xrightarrow{1} N_1, R_3 \xrightarrow{4} N_1 \}$; it also has three outgoing edges which can be depicted as $\{ N_1 \xrightarrow{0} R_1, N_1 \xrightarrow{0} R_2, N_1 \xrightarrow{0} R_3 \}$. After network N_1 is skipped over, the path $R_1 \xrightarrow{3} N_1$ is extended and transformed into two edges: $\{ R_1 \xrightarrow{3} N_1 \xrightarrow{0} R_2 \Rightarrow R_1 \xrightarrow{3} R_2 \}$ and $\{ R_1 \xrightarrow{3} N_1 \xrightarrow{0} R_3 \Rightarrow R_1 \xrightarrow{3} R_3 \}$. Similarly, edge $R_2 \xrightarrow{1} N_1$ is transformed into $\{ R_2 \xrightarrow{1} R_1, R_2 \xrightarrow{1} R_3 \}$, and the edge $R_3 \xrightarrow{4} N_1$ is transformed into $\{ R_3 \xrightarrow{4} R_1, R_3 \xrightarrow{4} R_2 \}$. Therefore in Figure 4-3, for router R_1, R_2 and R_3, there is a T-Edge between any two of them. Once all the single hop connections are formed, they will be taken as a normal edge in the transformed topology graph, with each edge actually standing for two directed edges with one in each direction, as explained in Section 4.2.1.1.

4.2.2.2 Choosing Tracking Method

The approaches to tracking OSPF behaviour, particularly network topology relevant behaviour, normally fall into three categories:

- **Periodic Dumping of OSPF Configuration Files**

  Intuitively, the network topology is formed as the result of configuring routers according to a pre-defined topology graph. By dumping configuration files available from each router, and parsing and analyzing the contained configuration data, an entire view of the network can be reconstructed.

However, this approach requires a thorough understanding of an IP router's operations. When errors exist in configuration data, these errors must be identified and removed. More importantly, this approach only provides a static view of the network topology. To make it more dynamic, the dumping frequency can be increased, but it is hard to go beyond a certain limit, due to the overheads caused by data collection.

- **Participating in OSPF Packet Exchange**

In this approach, the OSPF topology tracker acts as a dumb OSPF router attached to a real OSPF router via a point-to-point link. By doing so, an adjacency relationship can be fully or partially formed between the topology tracker and the router, such that the LSAs are exchanged between them [Shaikh04]. Here, *fully* refers to the adjacency being completely set up but special measures are taken to stop the topology tracker being used by other routers to forward IP packets (e.g. by assigning infinite OSPF weight on the links incoming to it, or setting up strict route filters on it). And *partial* here refers to keeping the process of setting up the adjacency failed to complete (e.g. including fake LSAs in Data Description packet but never actually sends out LSAs), such that the router will not be able to include a link to the topology tracker, but still keeps sending LSAs to it.

Since the topology tracker participates in LSA exchange, the advantage of this approach is that the link-state database can be reliably and quickly bootstrapped, and this mode can be used on any type of network media without constraints. However, the major problem of this approach is that it will impact the network, at least the attached router. As a result, the router might trigger SPT calculation too frequently if the topology tracker repeats setting up adjacency with it, and the router's memory will be greatly wasted in sending LSAs in vain.

- **Passive Snooping of OSPF Packets**

Since the OSPF packets are encapsulated in IP packets with their own protocol number, a protocol monitor can be deployed to passively snoop OSPF packets; thus the OSPF topology can be gradually formed as packets are captured. Due to its complete passive manner, this approach has the advantage of not introducing any network traffic. As a result, OSPF routers will not be aware of the protocol monitor's presence and will not be affected at all.

However, the drawback with this approach is that the topology is formed incrementally; that is, if no network topology change happens, one must wait for routers to refresh their LS database (e.g. normally 30 min). This is often undesirable for administrators

who wish to visualize a newly configured network.

As can be seen, there are pros and cons of each option discussed above. For the dumping option, it is error-prone, complicated, and only provides a relative static topology view. For the participating option, the overhead caused to the network is not desirable. Finally for the passive snooping option, although it is safe with low overhead, initial delay might be introduced when the system starts to capture the LSU packet. Given that OverMon is to be deployed on edge routers, the place where OSPF packets are initiated, the capture of LSU packets is very reliable. With the reliability of capturing OSPF traffic, as well as none overhead being introduced, the initial delay should be tolerable. As a result, OverMon chooses the method of snooping OSPF packet passively.

4.2.2.3 Identifying OSPF Packets

As discussed, at a high level, OSPF defines five types of packets, with each playing different roles in the OSPF routers' conversation and interaction. For example, the Hello packet can be used to bring up the adjacency relationship at the conversation starting stage; while the Database Description packet can be used to list the contents existing in the LS-database after the conversation is successfully setup; following that, the interaction stage starts by requesting link-state information (LS Request), exchanging link-state information (LS Update), and ensuring the link-state information is reliably received (LS ACK). As can be seen, topology information is encapsulated and exchanged by the Type 4 LS Update packet, while all others contribute to make this happen.

The link-state information contained in an LS Update packet is in the unit of LSA, which can be further divided into five types, namely router, network, summary-1, summary-2, and external-AS, each having a separate role in representing the network topology.

In Section 4.2.2.1, it has been clarified that the target topology graph in OverMon is the result of transforming the OSPF topology graph with sub-set elements, i.e. the vertices only consist of routers, not network segments; and the scope is within a single AS domain. Therefore, some types of LSA can be ignored. However, OSPF is complicated in that it uses different LSAs to describe different part of a network topology; it reuses the same data structure for different meanings of different network elements; and it maintains multiple copies of the LS database in describing different areas that a router belongs to. In other words, the LS-database is *incrementally* constructed and maintained; one edge in the OSPF topology graph might come from multiple different types of LSAs.

As a consequence, when an out-of-band application, such as OverMon, attempts to track

the OSPF packets for topology transformation, not only must it identify and capture the relevant packets, but also it must *correlate* the captured packets to produce a validated network element in the graph.

Therefore, examining the five types of LSA packet, **router-LSA** is the first to be retained, since it describes the routers and their working connections. On the other hand, **AS-external LSA** is the first type to be ignored, since it is used for the description of routes to destinations that are external to the AS, and these destinations normally are networks, not routers. Regarding **network-LSA**, although network vertices are not included in the OverMon topology graph, they can act as the transit point connecting routers; as a result, network-LSA packets *can not* be simply ignored. Lastly, in the case of **Summary-LSA** (i.e. both type 1 and type 2), the destination it describes is external to the area but still within the AS domain; the destination is either an IP network, an AS boundary router, or a range of IP addresses. Since a broadcast network enables OverMon to capture any OSPF LSU packet regardless the logical area division, the information contained in summary-LSA can be obtained from corresponding router LSAs or network LSAs, hence summary LSA can also be ignored.

In summary, to track OSPF topology information on a broadcast network with routers as the vertices only, OSPF LS Update packets (type 4) need to be snooped; router LSA and network LSA packets need to be partially processed, as well.

4.2.2.4 Parsing OSPF Packets

Each LSA packet begins with a standard 20-byte header, which contains three fields, namely *LS Type*, *Advertising Router*, and *Link State ID*. These three fields work together to identify a unique LSA packet. Among them, the field of *LS Type* uniformly dictates the type of a LSA packet; the field of *Advertising Router* uniformly specifies the OSPF Router ID of the LSA packet's originator; and the field of *Link State ID* in a router-LSA specifies the originating router's Router ID, while in a network-LSA it specifies the IP address of the network's *Designated Router (DR)*.

Here, the concept of *Designated Router (DR)* needs to be introduced. In broadcast and NBMA networks where a network segment is acting as a transit element allowing more than two routers to have interfaces to it, rather than setting up adjacencies between each pair of these routers, OSPF elects one of the routers as the DR for the network, and lets every other router establish a full adjacency with the DR. Then for a network segment, only the DR originates the corresponding network LSA, which contains the links to all its fully

adjacent neighbour routers in the network. By doing so, the link state information of multi-access networks is only governed by the DR; as a result, the protocol traffic and the size of the topological database can be greatly reduced. To provide additional resilience, OSPF routers also elect a Backup Designated Router (BDR), which becomes a new DR if the DR fails.

Next, to parse the topology information contained in LSAs, router-LSA and network-LSA are examined respectively; the detailed data structure of packet fields can be found in RFC 2328.

- **Network-LSA**

  The network LSAs play a key role in constructing the second type of edges in the OverMon topology graph, i.e. the *T-Edges* as defined in Section 4.2.2.

  In the OSPF routing protocol, a network-LSA is originated to describe a network segment. The network number of the network segment can be obtained by masking the value in the field of *Link State ID* (i.e. the IP address of the DR's interface on the network), with the value in the filed of *Network Mask*. This information is discarded by the transforming process though, since the network vertex does not exist in OverMon's topology graph. Rather, the field of *Attached Router* is useful, since two edges with one in each direction need to be constructed between each pair of the *Attached Router* (i.e. the RID of each attached router).

  Note that, by parsing network LSAs only, the *T-Edge* still can not be completely constructed, since network-LSAs do not contain any metric information; the absent information has to be obtained from relevant router-LSAs, as discussed below.

- **Router-LSA**

  The router-LSAs describe a router's working connections, i.e. the interfaces or links that can be envisioned as outgoing edges of a router in an OSPF topology graph.[8] Given that different types of physical networks are supported in OSPF, (e.g. point-to-point, broadcast, NBMA, virtual link, and point-to-multiple-point), they are treated by OSPF routers differently. That is, when necessary, OSPF decomposes a physical interface into one or more *link descriptions*, and accommodates these link descriptions into the associated router-LSA, with each in its own data structure to characterize the features of the link's destination (i.e. the address or ID), the type, and the cost etc.

[8] To be consistent with RFC 2328, from now on, "link" is generally used as the term for a router's connection.

For topology transformation, these link descriptions largely fall into two categories, namely router destined or network destined, depending upon whether the destination is a router (i.e. point-to-point link, and link to virtual link) or a network (i.e. link to stub network, and link to transit network).

- **Router Destined** For this category, a directed edge needs to be constructed from the *Advertising Router* (i.e. the originator of LSA) to the *Link ID* (i.e. the RID of the entity on the other end of the link), and with weight being set as the value encapsulated in the link description. In other words, this type of edge is constructed from genuine direct connections between routers, therefore it is typed as a *G-Edge*, as defined in Section 4.2.2.
- **Network Destined** For this category, since network vertices are not included in OverMon's topology graph, the type of *link to stub network* is discarded straightaway; whereas in the case of *link to transit network*, it contributes in constructing a *T-Edge*, i.e. it provides the weight information absent from T-Edge construction, when parsing the corresponding network-LSAs. In detail, when the network is transit (say Nt), each attached router R_{att} originates a router-LSA to contain the link description for its connection to Nt; the weight information (say w) contained in this link description stands for the cost value from the advertising router R_{adv} to the transit network Nt (for the other way around, from Nt to R_{adv}, the cost is always zero). When Nt is skipped over and a set of T-edges are constructed between any pair of attached routers, w is the weight value for those edges whose source is R_{adv} while its destination is one of R_{att} but not the R_{adv}.

Therefore, by snooping and correlating OSPF router-LSAs and network-LSAs, the two types of edges in OverMon's topology graph can be constructed.

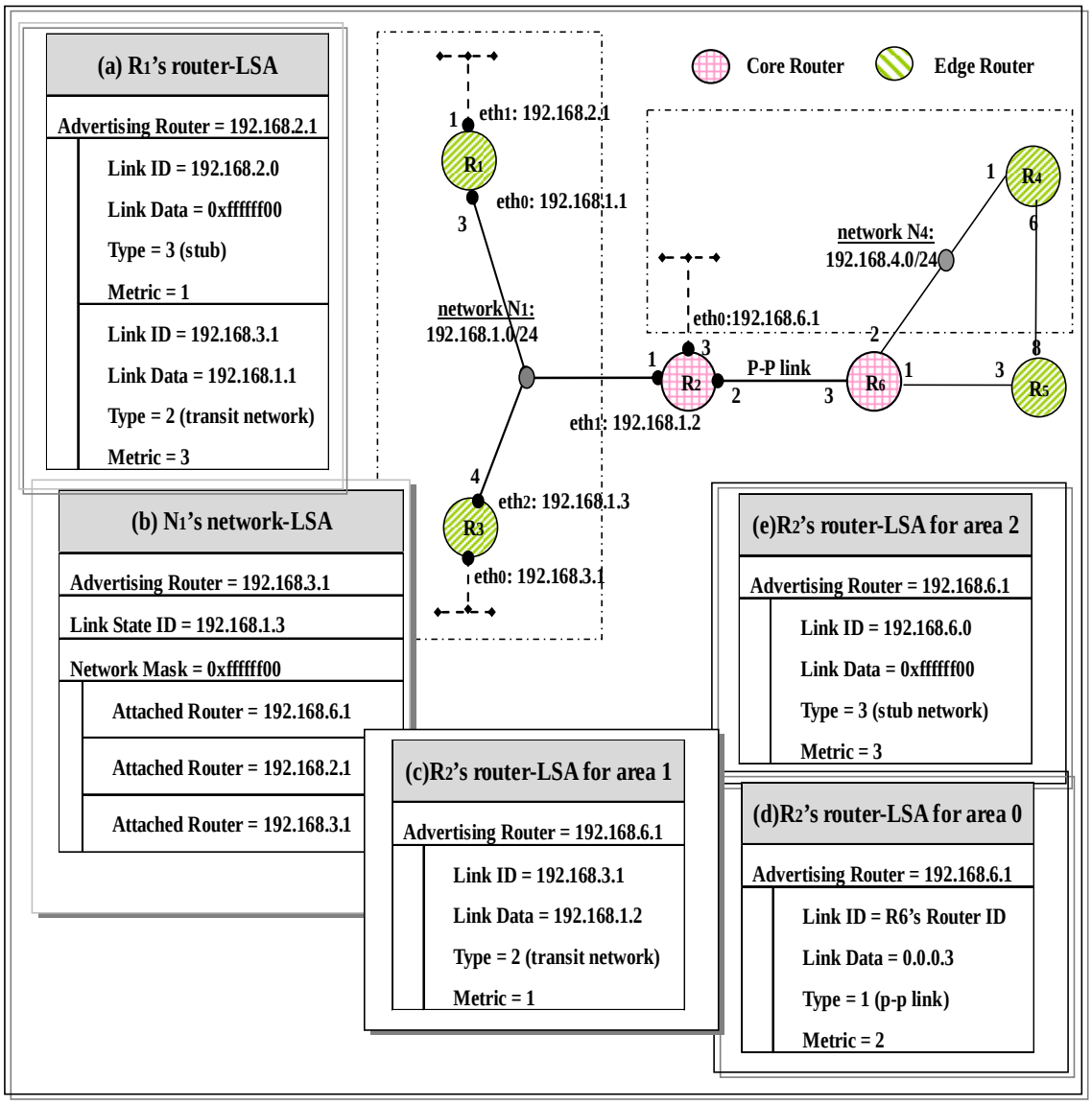


Figure 4-4: The resulting topology graph with associated LSAs

Figure 4-4 illustrates a resulting OverMon topology graph transformed from the previous OSPF topology graph shown in Figure 4-2. As can be seen, the stub networks N_2, N_3, N_5, and N_6 have been excluded; the transit networks N_1 and N_4have been skipped. The sub-table (b) shows network N_1's network-LSA, which is originated by its DR R_3, with RID equal to 192.168.3.1, (because R_3's RID is the highest). The RIDs of attached routers, such as R_1, R_2 and R_3 (i.e. the DR itself), are specified in the *Attached Router* field. Note that in Figure 4-4, core routers and edge routers are distinguished by different colours. For OverMon, core routers are defined as routers connecting different ASes, approximately equal to ASBR defined in OSPF; core routers are not deployed with the OverMon software,

hence do not participate in overlay construction, although they are still involved in routing IP packets as usual.

Lastly, a router-LSA is also used to indicate whether the router is an area border router (ABR). For an ABR, it originates a separate router-LSA for each area that it is attached to. In other words, if a router is an ABR belonging to multiple areas (say m), it originates m router-LSAs; otherwise if a router is an IR, it only originates one router-LSA. As shown in Figure 4-4, router R_2 is an ABR belonging to area0, area1 and area2; it originates three router-LSAs for each area, as shown in sub-table c, d, e respectively.

4.2.2.5 Maintaining Topology Graph

Due to the hop-by-hop based LSA flooding, an OSPF router may receive multiple copies of an LSA packet from different neighbouring routers. By passively capturing OSPF packets flying by on a broadcast network, the chance of receiving duplicate packets is even larger. To provide overlay construction with the topology graph in real-time, and at the same time efficiently use the resources at the edge routers, the topology graph should be recalculated only when the network topology changes, i.e. with an LSA record being added, deleted, or expired. To do so, similarly to the OSPF routing protocol, OverMon maintains a dynamic LS database, and converts it into a topology graph when required. Figure 4-5 shows the functional modules required for this purpose.

Starting from the left hand side, upon receiving an LSU packet that contains a set of LSAs, OverMon firstly performs a validation check against each LSA; thus only valid LSAs are further parsed. For example, besides the three fields (i.e. *LS type*, *Link State ID* and *Advertising Router*) in the header that uniquely identify a LSA instance, other fields (e.g. *LS type, LS sequence number, LS checksum)* are used to verify if the contents have been changed.

Once the LSA data is verified and parsed, the existence of an old copy of the same LSA instance will be checked.

- In the case of the incoming LSA being new to the database, it is appended to the LS-database; at the same time, an age counter starting from zero is associated with it. Next, the age counter enters a queue which is ticked by a timer every one minute. To age out the LSA records that reside in the link state database, if a LSA record's age count reaches to 31, i.e. if it exists in OverMon's LS database for longer than 30 minutes, it will be expired and deleted, which leads to the topology graph being recalculated. Note the 30 minute setting is consistent with the default refresh-

period in OSPF.

- In the case that an old copy exists in the database, the comparison of freshness is further performed: if the coming copy is less fresh than the old copy, the LSA record will be refreshed with the age count being reset to zero; otherwise, the old copy will be deleted and the new copy will be added into the database, Note that if in the incoming LSA, the field of *LSA Age* is set as *MaxAge*, the old copy is deleted from the database directly, as that value indicates that the LSA is no longer valid.

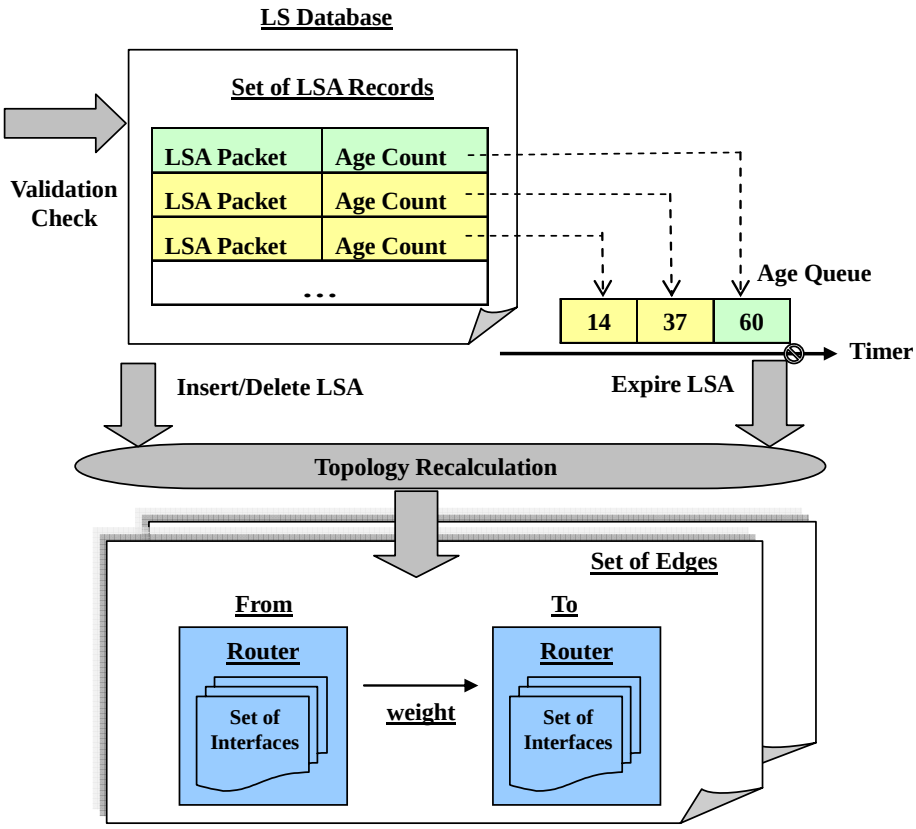


Figure 4-5: The functional modules in maintaining and representing OverMon topology graph

The resulting topology graph is in the form of a set of directed edges, with each consisting of three components, namely *from*, *to* and *weight*. The *from* and the *to* stand for the two ends of an edge, which are OSPF routers configured with a list of interfaces; the *weight* stands for the weight value associated with the edge.

With this topology graph, overlay networks can therefore be constructed as described in the following sections.

4.3 Control Overlay

The control overlay is used by OverMon to multicast the control information, i.e. the parameters of measurement tasks that are to be initiated, to the participant nodes.

With the analysis in Section 3.2.4, the multicast problem in OverMon can be characterized as being *lightweight*, since the membership for each multicast session is static and pre-known, and the period that a session lasts is short. In other words, it differs fundamentally from other popular application level multicast applications. More importantly, with network topology information available, a topology-aware multicast overlay can be constructed, which can improve the performance without causing any maintenance traffic.

In this section, the construction of the control overlay in OverMon is discussed. The problem to be solved is firstly formalized, then the tree construction algorithm and ALM routing algorithm are presented respectively. In Chapter 6, the performance of these algorithms will be examined.

4.3.1 Problem Formalization

Multicast is a problem to which a tree data structure is typically applied; a multicast delivery tree is rooted at the source node and the contents are delivered along the tree to the multiple receiving nodes. Briefly, when an undirected graph is denoted as $G = (V, E)$, where $V = \{v1, v2, ...vN\}$ is the set of vertices in G, $E \subseteq \{\{vi, vj\} | vi \in V, vj \in V, and\ i \neq j\}$ is the set of edges in G, a *tree* of G is a connected sub-graph of G such that the removal of any edge in the sub-graph will make it disconnected; when the tree spans V, it is a *spanning tree* of G [Kou81].

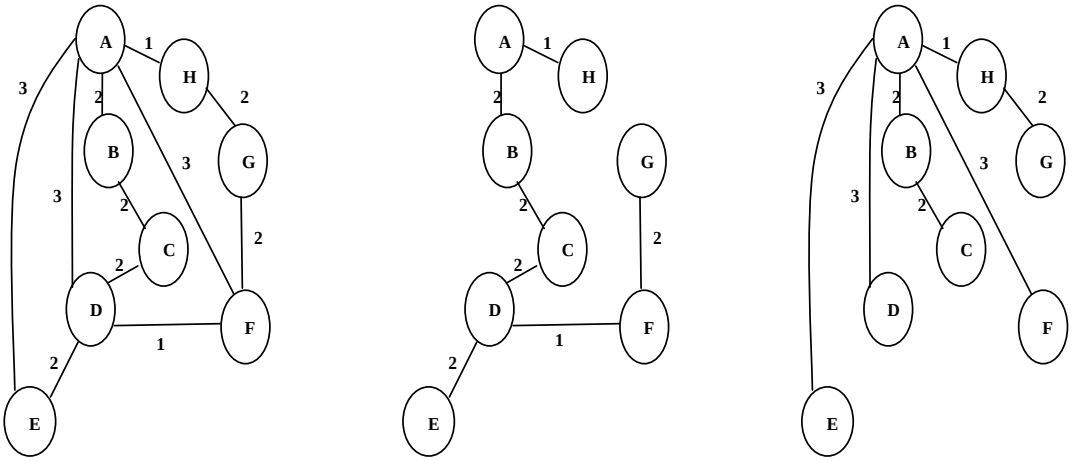
Such a delivery tree can be optimized from two strategies: one is to achieve minimal total tree cost; the other is to achieve minimal pair-wise cost, i.e. end-to-end delay. Accordingly, to achieve minimal total cost, a ***minimum spanning tree (MST)*** is often the solution, in which a spanning tree of G is constructed such that it has minimal total distance on its edges among all spanning trees of G. On the other hand, to achieve minimal end-to-end delay, the ***shortest-path tree (SPT)*** if often the solution, in which a tree of G is constructed such that the distance between any vertices in the tree to the root vertex is minimum. When the receiving node set S is a subset of node set V, the smallest tree connecting all the vertices of S is called a ***Steiner Tree (ST)***. The ST problem is distinguished from the MST problem in that it is permitted to construct or select intermediate connection points to

reduce the cost of the tree [Kou81]. Particularly, the ***Steiner Minimal Tree (SMT)*** problem has been studied in several papers [Garey79, Gilbert68, Winter87, Kou81, and Zelikovsky93]. According to the literature survey, although an SMT provides the optimal-cost multicast tree, its computation is NP-complete [Garey79]; therefore many existing heuristic algorithms provide good approximation of SMT within a small network size [Pendarakis01].

The applications to which MST and ST are largely applied include network design, wiring layout, and tour planning; while SPT is widely applied to IP multicast or other applications (e.g. for broadcasting services) where the target is to minimize the network delay. Figure 4-6 illustrates an example of building different types of multicast trees based on a same physical network topology.

In the case of OverMon, firstly, the multicast tree construction problem is a ST problem, since OverMon is only deployed on edge routers, not on core routers, thus the receiving nodes, together with the root node, on the multicast tree, are a sub-set of all vertices in the topology graph. Secondly, since core routers are not involved in multicast tree construction, i.e. core routers are not aware of OverMon's existence, Steiner tree construction in OverMon needs to specifically exclude the core routers. Thirdly, regarding the optimization of tree construction, as discussed in Section 4.1.1, the priority for ALM construction in OverMon is to achieve minimal total bandwidth cost, rather than the minimal end-to-end delay cost. In summary, the multicast tree construction in OverMon can be formalized as follows.

Given that there exists a unicast path between any pair of routers, the underlying physical network can be modeled as a complete graph $G = (V, E)$, where V is a set of (core or edge) routers and E is a set of directed links between pairs of routers. Among V, Re is the set of edge routers while Rc is the rest set of core routers, i.e. $V = Rc + Re$. Given an initiating node $r \in \mathrm{Re}$ and a set of other nodes $r' \in \mathrm{Re}$, a Steiner tree rooted at r covering r' form a set of overlay paths, on which the monitoring task parameters are disseminated; the total bandwidth cost of tree construction should be as low as possible



(a) Graph (b) Minimal Spanning Tree (c) Shortest Path Tree

Figure 4-6: Different multicast trees built upon the same topology graph

4.3.2 Tree Construction Algorithm

With the network topology available from OSPF snooping, the Steiner tree construction in OverMon is straightforward; the algorithm that is used in OverMon mainly evolves from the heuristic algorithm H [Kou81]. It consists of five steps as follows; Figure 4-7 illustrates an example of the original graph and the resulting Steiner tree.

Algorithm H

INPUT: an undirected distance graph $G = (V, E, d)$ and a $_{set}$ of Steiner points $S \subseteq V$

OUTPUT: a Steiner tree T_n for G and S.

Step 1: Construct the complete undirected distance graph $G_1 = (V_1, E_1, d_1)$ from G and S such that $V_1 = S$ and, for every $\{v_i, v_j\} \in E_1$, $d(\{v_i, v_j\})$ is set equal to the distance of the shortest path from v_i to v_j in G.

Step 2: Find the minimal spanning tree T_1 of G_1- If there are several minimal spanning trees, pick an arbitrary one.

Step 3: Construct the sub-graph, G_s, of G by replacing each edge in T_1 by its corresponding shortest path in G - If there are several shortest paths, pick an arbitrary one.

Step 4: Find the minimal spanning tree, T_s, of G_s - If there are several minimal spanning trees, pick an arbitrary one.

Step 5: Construct a Steiner tree T_n from T_s by deleting edges in T_s, if necessary, so that all

the leaves in T_n are Steiner points.

Note that the reconstructed topology graph in OverMon, as discussed in Section 4.2, is a directed graph. To apply this algorithm, the directed graph is transformed into a un-directed graph by unifying two directed edges into one, and resetting the weight value of each undirected edge as the sum of the two directed edges, i.e. $w_{undirect} = w_{direct\_out\_going} + w_{direct\_in\_coming}$. For the algorithm that is used to construct the shortest path as required in Step 1, and for finding a minimal spanning tree as required in Step 2 and Step 3, well known algorithms such as [Dijkstra59, Floyd62, Tabourier73, Yao75, and Cheriton76] can be used.

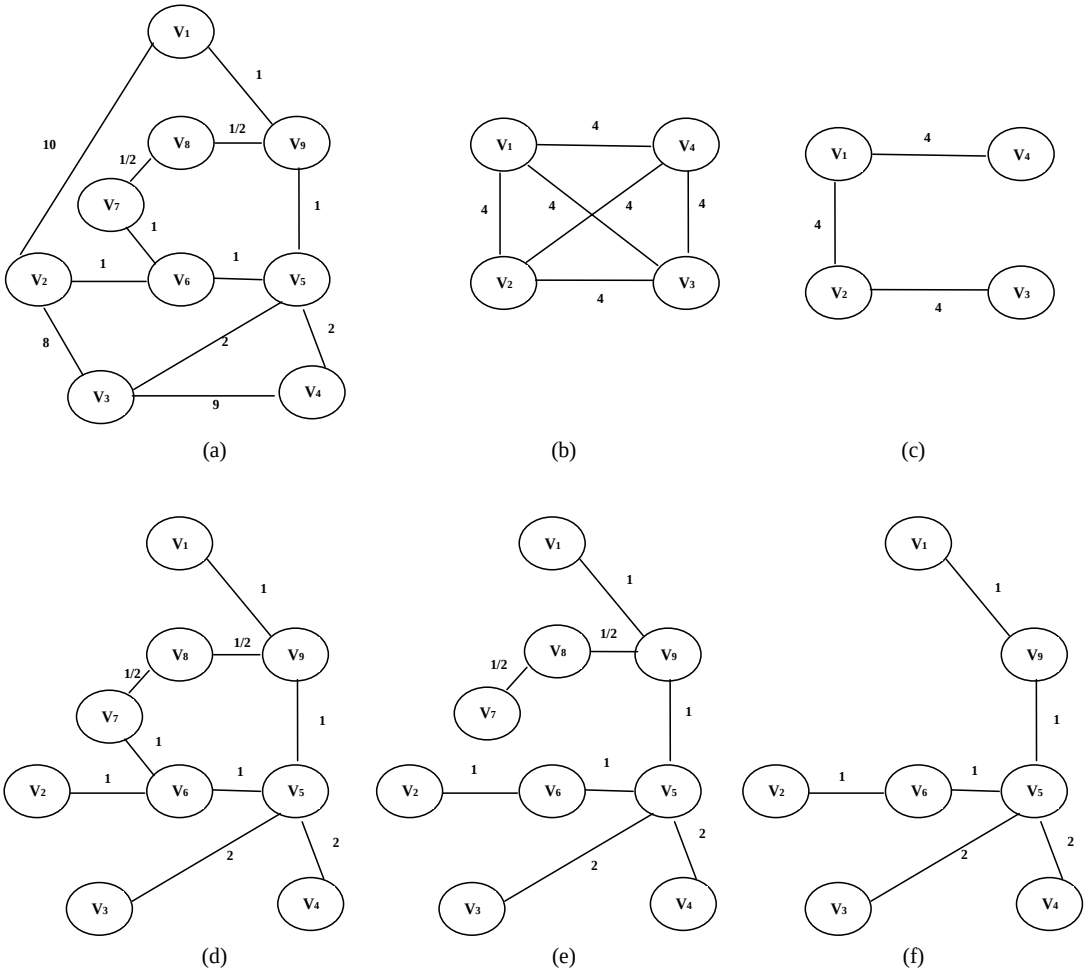


Figure 4-7: A sample execution of algorithm H [Kou81]

With regards to computational complexity, by using these mentioned algorithms, in the worst case, Step 1 could be completed in $O(|S||V|^2)$ time, Step 2 could be completed in

$O(|S|^2)$ time, Step 3 could be completed in $O(|V|)$ time, Step 4 could be completed in $O(|V|^2)$ time, and Step 5 could be completed in $O(|V|)$ time. Overall speaking, Step 1 dominates the computational time; hence for a given $G = (V, E, d)$ and $S \subseteq V$, the worst case time complexity of the heuristic algorithm is $O(|S||V|^2)$. In addition, it has been proved that the Steiner tree produced by the algorithm H is not minimal; however, Dh , the total distance on the edges of the Steiner tree is not very far from $D\min$, the total distance on the edges of the minimal Steiner tree, $Dh / D\min < 2(1 - \frac{1}{l})$ where l is the number of leaves in the minimal Steiner tree[Kou81].

With respect to OverMon, the distributed version of algorithm H is applicable except steps 3-5 are not necessary. This is due to the fact that, in the algorithm H, network nodes are equally considered, in the sense that although the generated tree only takes task parameter receiving nodes as the leaf nodes, the internal nodes[9] of the tree might contain non-receiving nodes. This is not acceptable for OverMon, since in OverMon, network nodes are grouped by edge routers and core routers, and OverMon is only deployed on edge routers. As a consequence, if core routers are on the path of the Steiner tree, the tree construction messages will not be processed by core routers, such that the multicast tree will not be successfully constructed.

Essentially, the most important step of algorithm H is Step 1, which constructs a complete undirected distance graph $G1$, of which, the vertices are the set of task participant nodes, and the path between any two vertices is the shortest path between them as they are in graph G .

Given that the up-to-date network topology information can be re-constructed by the topology tracker, it is safe to assume that although $G1$ is calculated at the application level at run-time, the paths on $G1$ are *almost* identical to the shorted paths calculated by the OSPF routers at the network level; as a result, the spanning tree generated from $G1$ in Step 2 is very close to the minimum spanning tree calculated by an OSPF router at network level. Here, by *almost*, it refers to the fact that the edge routers and the core routers are quite stable; with a very large probability, the topology graph re-constructed by OverMon nodes at the application level are *identical* to each other, and are identical to the real network topology.

In summary, with the same level of computation complexity as algorithm H, by taking Step

[9] An internal node is any node of a tree that has child nodes and is thus not a leaf node.

1 and Step 2 of algorithm H, a Steiner tree overlay in OverMon can be efficiently constructed on edge routers, with the given node that initiates the task as the root node, and task participant nodes as the internal nodes and leaf nodes. The evaluation of this approach is discussed in Chapter 6.

4.3.3 ALM Routing Algorithm

In this section, the ALM routing algorithm is discussed; it is the protocol that is used by OverMon nodes to communicate and cooperate for the dissemination of task configurations.

Generally speaking, to construct a distributed multicast tree, there are two types of approaches, namely ***source-tree*** based, e.g. DVMRP [Waitzman88] and ***shared-tree*** based, e.g. CBT [Ballardie93] and PIM [Deering96][10].

Typically, the source-tree based routing algorithm applies separate multicast trees for each sender. To build a good tree with low cost (e.g. low latency or low bandwidth consumption), network link state information needs to be collected, and group membership needs to be flooded to all group members. This may overwhelm the network nodes and produce the predominant overhead for the networks. When the network is in a complicated and heterogeneous environment, such as the Internet, the processing cost of tree calculation is also a drawback. Thus, source-based approach might not scale well in a large network.

On the other hand, the shared-tree based approach can achieve better scalability and simplicity by creating a shared tree rooted at the Rendezvous Point (RP). With regards to scalability, it is particularly favourable in a network with rich connectivity and a large number of sources. However, the main drawback of using a shared-tree in a network is that *traffic concentration* is likely to be caused: if each sender uses the same shared-tree, the traffic might become congested along certain links of the shared-tree. Another drawback of the shared-tree is that the sender and the receiver may not be connected by the shortest path; hence the end-to-end delay could be higher than the source tree routing counterpart.

In the case of OverMon, as discussed in Section 3.2.4, it is desired to be completely distributed, and any edge router within the network can be chosen to initiate a measurement task; as a result, multicast sessions with different source nodes and receiver nodes might exist concurrently. The shared-tree based approach, although it scales better in very large

[10] PIM supports both shared-tree and source-tree: when the members are densely distributed, PIM uses share-tree; while when members are sparsely distributed, source-tree is used.

network, from each session's perspective, the overall cost might not be minimal. Additionally, given that the membership of each ALM session in OverMon is pre-known, and the up-to-date topology can be tracked at each edge router where OverMon is deployed, the overhead in maintaining source trees, as discussed above, imposes little traffic on the network. Therefore, the source-tree based approach is adopted.

The ALM routing algorithm in OverMon is simple in that only the root node calculates the Steiner tree; all other nodes just relay and respond to the overlay construction messages by following the generated tree. The details are as follows.

- **Root Node**: The root node is an edge router running OverMon software, and chosen to initiate a measurement task. It receives a task parameter specification which contains a list of nodes that are required to participate in the task, i.e. to receive the ALM packets. With this receiving node list, the root node calculates the Steiner tree based on the up-to-date topology provided by its *Topo.Tracker* component as discussed in Section 4.2. The vertices on the tree only include receiving nodes, and the root node could also be one of the receiving nodes. The generated tree is then encapsulated into a tree data structure that contains the list of all nodes including the root node. In this data structure, each non-root node has an associated pointer pointing to its parent; the root node's parent pointer is null. This tree is encapsulated into a multicast packet and sent to the direct children of the root node. The root node also sets up a timer while waiting for these children's responses. When all responses for which it is waiting are returned, or when the timer expires, the root node returns the result, which also includes its own answer, to its client via an RMI interface.
- **Internal Node** An internal node is an edge router running OverMon software and has dual roles of receiving multicast packet from its parent, as well as relaying multicast packet to its children. On receipt of a multicast packet, it firstly decodes the multicast packet into a multicast tree data structure. Then, it finds out its children according to the received tree, and relays the whole tree to its children. Similar to the root node, it also sets up the timer waiting for its direct children's responses; differently, it returns the result packet to its parent, not the client. The result packet includes its own answer on whether the new task can be registered successfully.
- **Leaf Node** A leaf node is an edge router running OverMon that only has the role of receiving the multicast packet from its parent. It returns the response to its parent

immediately, with the answer of whether the task is accepted or denied, in terms of whether the task has any conflict with already registered tasks.

Note that in this algorithm, except at a leaf node, the response returned from an internal node to its parent is an accumulated answer: if an internal node's children all respond within the time out period, and all children plus itself respond with the answer of *accept*, it will respond to its parent with *accept*; otherwise, it will respond to its parent with *reject*, along with a list of nodes who either answered with *reject* or failed to respond within the time out period. Thus, when an internal node receives a *reject* answer, it will escalate the attached rejecting node list to its parent. This process recursively continues until reaching the root node.

One may argue that this accumulating style response is not scalable for large size multicast groups, in terms of the latency it introduces at the *upper* part of the tree. However, for this non-delay sensitive intra-domain multicast service in a relative stable environment, with careful software design and implementation (e.g. by setting up a timer thread dedicated to monitoring the asynchronous response), the latency should be tolerable and acceptable; as the latency is traded for the simplified calculation at edge routers, and splits the responsibility of processing replies at the root node to all descendant non-leaf nodes.

4.4 Data Overlay

The Data Overlay is an overlay-based data management solution employed by OverMon, to store measurement results, and to support distributed multiple-attribute range query.

In Section 3.3.2, overlay based data management solutions have been discussed, mainly from overlay technology's viewpoint. The focus there was to present a broad introduction to different data management solutions that are based on overlay networks, and their applicability to OverMon. In this section, the focus is multi-attribute range query. As discussed in Section 3.3.2.3, among the research efforts that are addressing this issue, Mercury, a protocol supporting multi-attribute range query, seems to offer what is required by OverMon. Thus, in this section, the problem of multi-attribute range query is briefly formalized in Section 4.4.1; then, the Mercury protocol is introduced in Section 4.4.2, followed by a discussion of improvements that can be made to Mercury in Section 4.4.3.

4.4.1 Problem Formalization

The multi-attribute range query problem can be viewed as a d-dimensional range query problem. Firstly, starting from 1-dimension range query, the problem can be formalized as below: Let $P := \{p_1, p_2, ... p_n\}$ be the given set of points on a line, a range query asks for the points inside a 1-dimensional interval $[a, b]$.

When P is a set of n points in the plane, without two points having the same x-coordinate and y-coordinate, a 2–dimensional rectangular range query is composed of two 1-dimensional sub-queries, one on the x-coordinate of the points, and the other on the y-coordinate. A 2-dimensional range query on P asks for the points from P lying inside a query rectangle $[a_x : b_x] \times [a_y : b_y]$. A point $p := (p_x, p_y)$ lies inside this rectangle if and only if

$$a_x \le p_x \le b_x \text{ and } a_y \le p_y \le b_y$$

Fairly straightforward, when P is a set of points in d-dimensional space, a point p in P is an ordered d-tuple $(x, y, ...d)$, a d-definitional range query on P asks all points from P lying inside a d-dimensional hyper-rectangles $[a_x : b_x] \times [a_y : b_y] \times ... \times [a_d : b_d]$. A point $p := (p_x, p_y, ... p_d)$ lies inside this hyper-rectangles if and only if

$$a_x \le p_x \le b_x \text{ and } a_y \le p_y \le b_y \text{ and } \ldots a_d \le p_d \le b_d$$

The d-dimensional range query problem originally is studied by the database community. The classic data structure addressing 1-dimensional range query can be a balanced binary search tree; other well known data structures addressing d-dimensional range queries include Kd-Trees, Range Trees, and Segment Trees.

Essentially, when this problem is extended into a distributed version, a distributed data structure is formed on the nodes of the network, i.e. the d-dimensional data space is partitioned among the network nodes, with each hosting one or multiple points in the data space. To support range query, each note constructs and maintains several links to other nodes, e.g. remote nodes with different IP addresses, herein the query can be routed to the node that is responsible for the data that is queried. The challenge is to design an efficient distributed data structure that leads to communication overhead and processing delay as low as possible.

4.4.2 The Mercury Protocol

The Mercury protocol is targeted at the problem of distributed range query on multiple attributes. Its essential features and applicability have been briefly discussed in Section 3.3.2.3. In this section, some part of its design directly relating to OverMon is discussed; more details can be found in [Bharambe04].

4.4.2.1 Routing Data Items and Queries

Mercury organizes attributes into hubs; each hub is a circular overlay with data being placed contiguously on the ring; each node is responsible for a contiguous range of values for an attribute. Within a hub, the routing algorithm is as follows:

Let neighbor n_i be in charge of the range $[l_i, r_i)$. When a node is asked to route a value v, it chooses the neighbour n_i for which $d(l_i, v)$ is minimized, where d denotes the clockwise distance between two nodes a and b along the ring. Let $\min_a$ and $\max_a$ be the minimum and maximum values for attribute a respectively, then

$$d(a,b) = \begin{cases} b-a & if\ a \le b, \\ (\max_a - \min_a) + (b-a) & if\ a > b \end{cases}$$

As a distributed data structure supporting range query, Mercury is simple in that:

- For a particular attribute, the range query can be routed to the *first* node covering the value appearing in the range being queried, and then, it uses the contiguity of range values to spread the query along the circle as needed.
- Hubs can be thought of as orthogonal attribute dimensions, so that conjunctive range query over multiple attributes can just be decomposed into multiple 1-dimensional sub-queries for each hub;

Figure 4-8 illustrates a routing example of Mercury. It depicts two logical hubs H_x and H_y, both having minimum value 0 and maximum value 320, representing attribute X and Y respectively. There are seven nodes $\{a, b...g\}$, of which, nodes $\{a, b, c, d\}$ participate in the whole range of $[0,320]$ for H_x evenly, and nodes $\{e, f, g\}$ participate in the whole range of $[0,320]$ for H_y unevenly[11].

In Mercury, a data item D is represented by a set of typed name-value pairs of attributes,

[11] Actually, a physical node in the system can be part of multiple logical hubs, depending on the start-time configuration.

i.e. a tuple of the form: $(type, name, value)$, while a query Q is a conjunction of predicates which are tuples of the form $(type, name, operator, value)$. The *operator* can be a general relational operator such as $\leq, \geq, \neq$. Mercury chooses to send data items to all the logical hubs that correspond to the attributes presented in the data item, while the query is sent to exactly one of the hubs that correspond to the attribute being queried.

The routing mechanism is described as follows. A data item within each hub is routed to the responsible node; the value of the item for that hub's attribute falls into the range of this node. While for a query, the first routing hop determines which hub to route to, the rest of the routing is completed within a chosen hub based on the values of that single attribute of the data item; and is delivered and processed by all nodes that might potentially have the matching values.

As the example depicted in Figure 4-8, the data item is sent to both H_x and H_y, and stored at nodes b and e respectively; the query enters H_x from node d first, it is then routed to nodes b and c for processing.

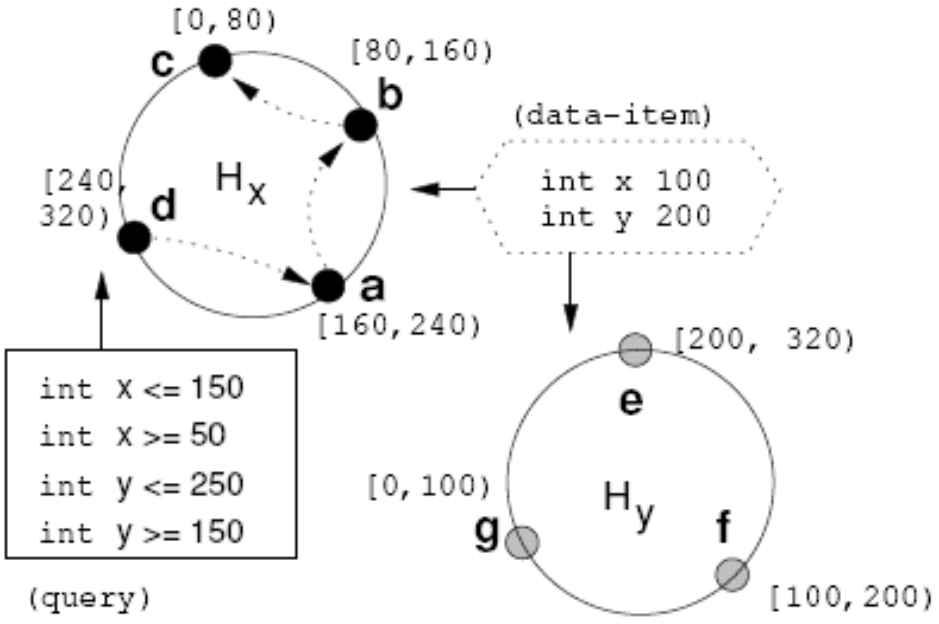


Figure 4-8: Routing data items and query in Mercury [Bharambe04]

4.4.2.2 Overlay Construction

Essentially, for overlay based data management, the process of overlay construction is the process of building up a distributed data structure; the focus is to establish and maintain the pointers that connect to other related nodes.

As seen above, for each attribute, Mercury uses a ring as the distributed data structure. That is, the whole range of an attribute is mapped to a virtual ring. Each node in the system, following the clockwise direction, takes the responsibility of a continuous part of the ring. To guarantee the correctness of routing data items and queries within a hub, two important features need to be maintained: connectivity and no-overlap, particularly when a node is responsible for a range that is starting from a value smaller than the maximum value of the whole range, and ending at a value larger than the minimum value of the whole range (due to the clockwise direction).

Therefore, the circular overlay construction for each hub can be imagined as building a distributed double-circularly-linked list: each node has *previous* and *next* links pointing to the closest neighbor in each direction; the *previous* link of the first node points to the last node, and the *next* link of the last node points to the first node. Correspondingly, for each node, it maintains two links to two immediate neighbors, one at each direction; these two immediate neighbors are named as *successor* and *predecessor*.

Besides having two links to the predecessor and successor within its own hub, each node also maintains a link pointing to at least one representative node of each of the other hubs in the system, and serves as the cross-hub link in query routing. Obviously, given that most of the routing occurs within an attribute hub, and the number of hubs is often low, the cross-hub link is not a significant burden to be maintained.

Thus far, simply using the successor and predecessor links can already ensure routing correctness. To optimize, i.e. to accelerate the process of finding the *first* value appearing in the range being queried, it is useful to maintain some extra links by which a shortcut can be setup directly to the node whose range is closest to the range that is being queried. Therefore, besides the two links pointing to its predecessor and successor, each Mercury node maintains a special type of long-distance link that points to a node far from the local node. The discussion of how these long-distance links are constructed is deferred until the next subsection. Rather, this section continues to discuss the protocol used by nodes to join and depart.

Node Join Like most other distributed overlay techniques, before a new node joins the system, it needs to know at least one node that is already part of the system. The incoming node queries this existing node and obtains state about the hubs, along with a list of representatives for each hub in the system. Then, it randomly chooses a hub to join and contacts an existing node m of that hub. The incoming node installs itself as a predecessor of m, to become a part of the hub. That is, it takes all the successors of m as its own

successors and half of $m's$ range as its initial range; then it initiates other maintenance processes, such as long-distance link construction, successor list maintaining etc.

Node Departure When a node departs, other related nodes need to repair their links, particularly the successor and the predecessor links that directly impacts the correctness of routing. To repair successor/predecessor links within a hub, a short list of contiguous nodes further clockwise on the ring than its successor is maintained by each node. Each node then pings the nodes in the list periodically to update their liveness, as well as the range information for which these nodes are responsible. Thus, when a node departs, it is its predecessor's responsibility to find the next node along the ring as its new successor. As to long-distance links, which is important for routing optimization, they are repaired in the next round of periodical reconstruction, by using the node count that is newly estimated (as discussed in the next section). Finally, to repair the broken cross-hub links, a node can either use a backup cross-hub link, or to query its successor or the predecessor for their links to the desired hub, or to query a bootstrap server (i.e. a node dedicated for processing bootstrap requests) for the address of a node participating in the desired hub.

4.4.2.3 Long-distance Link Maintenance

In this section, the establishment of long-distance links is discussed. Note that since most of the routing and link maintenance occur within a single hub, unless stated otherwise, the discussion in this section is based on a single hub, and n denotes the number of nodes within a hub.

Unlike the successor/predecessor links that are the key to ensure the correctness of routing, long-distance links are the key to ensure the efficiency of routing: by long-distance links, two distant nodes are connected such that routing steps can be shortened by taking shortcuts.

The long links can be constructed following this rule: a node A generates an integer $x \in (0, n)$ using the harmonic distribution $h_n(x) = 1/(n \log x)$, and establishes a link to the node B which is x links away in the clockwise direction from A. Based upon Kleinberg's analysis of small world networks [Kleinberg00], it can be proved that if each node constructs one such long link, the expected number of hops for routing to any value can be reduced from $O(n)$ to $O(\log^2 n)$. Furthermore, based on the work of Symphony [Manku03], if each node has k such long links, the routing hops can further be reduced to $O\left(\frac{1}{k}\log^2 n\right)$, k is configurable and can be different from node to node.

However, this result holds only under the assumption that the value ranges on each node are uniformly distributed. Otherwise, it is difficult and expensive ($O(x)$) for a node A to determine which node B is x hops away from itself.

Mercury deals with this issue by maintaining an approximate mapping of hop-count along the ring to the value-range assigned to nodes. That is, network nodes in Mercury sample others to determine how large a range they are responsible for; the samples are then used to create an estimate of the numbers of nodes in any part of the hub, i.e. a histogram of the distribution of nodes.

Once the histogram is formed, long-distance links are therefore established as follows. Firstly, the number of nodes n within a hub in the system is calculated. Then, for the long-distance link from node A to node B, a value n_l between $[1, n]$ is randomly generated by A using the harmonic distribution, representing the number of nodes that must be skipped along the circle in the clockwise direction to reach to B. The histogram is then used to estimate a value v_l for which the node B is responsible. Finally, a join message is sent to this value v_l from A, and is routed to B using the routing protocol discussed in Section 4.4.2.1. In Mercury, $k = \log(n)$ intra-hub long-distance links are suggested to be established by each node.

4.4.2.4 Random Sampling for Histogram Maintenance

As mentioned, sampling is to provide information for system level estimation, e.g. node-count distribution, routing load distribution, etc. The basic idea is as follows: each node performs *local estimation* first, i.e. estimate system statistics based on the information from local and near neighbours; these local estimates are then exchanged throughout the system using the sampling algorithm; since the exchanged estimates can be seen as points on the required distribution, by organizing them into an array, a piecewise linear approximation, i.e. a histogram, can therefore be computed.

To compute a high quality histogram, it requires the sampling algorithm to produce a *uniform* random sample of the nodes in the system. It has been proved in Mercury that the hub overlay is an *expander graph* with a high probability. An expander graph is a sparse graph which has high connectivity properties [Hoory06]. It also has the property that random walks over the links of such a network converge very quickly to the stationary distribution of the random walk. Since the hub overlay graph is regular, the stationary distribution is the uniform distribution.

With this guarantee, a simple epidemic style random-walk based sampling algorithm can

be performed as follows: nodes send off a sample-request message with a small Time-To-Live (TTL) to a randomly chosen neighbour; the neighbour decrements the TTL and forwards it to its neighbour in a similar way; the node at which the TTL expires sends back a sample to the original sender.

Recall that a long-distance link is established with node count n as a parameter, and n can be calculated by a histogram estimating the number of nodes in any part of the hub; let's take node-count as an example to explain how this mechanism for maintaining histogram using random sampling works.

- Firstly, a *local estimate* is made by each node. That is, let N_d denotes the *local* d-neighbourhood of a node – i.e. the set of all nodes within a distance d ignoring the long-distance links, each node periodically queries the nodes $\in N_d$ and produces a local estimate of the system statistic under consideration. In the case of node-count, a node's local estimate can be $(\max_a - \min_a)|Nd|/(\sum_{k \in N_d} |r_k|)$ where r_k is the range of a node k, and $\min_a$ and $\max_a$ are the minimum and maximum values for attribute a; it represents an estimate of the node-density as a point on the hub.

- Next, these local estimates are exchanged as samples using the sampling algorithm. That is, a node periodically samples k_1 nodes uniformly at random using the sampling algorithm; each node reports back its local estimate, as well as the most recent k_2 estimates it has received. As time progresses, a list of tuples of the form: $\{node\_id, node\_range, estimates\}$ can be built up by a node; they can be utilized to compute the histogram of the system statistic that is being measured. Note that, k_1 and k_2 are the parameters of trading sampling overhead with accuracy of the histogram maintenance process, and $\log(n)$ is suggested to be the reasonable setting.

- Finally, the histogram is computed. If it is to generate an average or histogram of *node* properties, the collected samples can be used exactly as they are collected. Otherwise, to generate an average or histogram of properties *around the routing hub*, some minor modifications are needed. For example, node-count for the hub can be computed by summarizing the values of each point on a histogram that represents the number of nodes in any part of the hub; this histogram is formed by stitching the samples that represent a node-density for different points on the hub. In order to generate unbiased node-count histograms, the received samples are weighted differently: samples reporting lower densities are given higher weight to account for the fact that, there would be fewer nodes to produce low density

samples.

In addition to node-count, other system level statistics can be estimated, such as load balancing. That is, according to the histogram, average load in the system can be calculated, such that a node can determine whether it is relatively overloaded. A heavily loaded node can then send probes to lightly loaded part of hub; once the probe is received and responded by a lightly loaded node, the lightly loaded node will gracefully leave its old location and rejoin at the new location of the heavily loaded node.

4.4.3 Topology-Aware Overlay Construction

From the discussion thus far, Mercury can be applied as a building block for OverMon to construct its data overlay. Given that network topology can be obtained through the topology tracker as introduced in Section 4.1, it is interesting to investigate how this special information can be leveraged to improve the performance of Mercury; and even more, to be a general topology-aware data overlay solution.

This section discusses this issue by firstly analysing the space left for this improvement in Mercury; then presenting a heuristic algorithm that is designed for this purpose.

4.4.3.1 Space for Improvement

Mercury does not consider network proximity. That is, a new node randomly chooses an existing node to join the system, without considering network level distance between them; the constructed overlay might result in one hop along the ring actually consisting of multiple continuous and duplicate links on the underlying physical network. This unavoidably will consume more network bandwidth.

In addition, to achieve load balancing, Mercury's approach of moving a node around in the ring is a widely accepted solution [Godfrey04, Byers03, Karger04, and Ganeshan04], particularly suitable to address the issue of *execution skew* [Ganeshan04]. By doing so, the workload is evolving over time hence they can adaptively deal with variant load distribution. Something in common with these approaches is that, this *leave-join* mechanism unavoidably introduces movement cost in both data and node movement (i.e. it involves sending $\log(n)$ probes in parallel, each of which must traverses $1+\log(n)$ hops [Bharambe04]), and cause complication in setting up the cost model and the threshold value, and consideration of the heterogeneity of nodes etc. Due to the complicated nature of this topic, OverMon does not intend to address it extensively in this dissertation. However,

it can be observed that the initial range that each node is assigned to when it firstly joins the network, plays an important role in routing, as the routing load that each node experiences could relate closely to the range for which it is responsible, especially if nodes' membership is quite stable as in OverMon.

Based on the analysis above, the overlay construction algorithm of Mercury can be enhanced, to make it topology-aware, at the same time to minimize load imbalance. The reason behind this argument is: rather than choosing a random node to join, a node can choose a node that is nearest to itself at network level, hence to reduce the network level hops that one overlay path actually covers. Furthermore, this decision can be made as a compromise between least hops and balanced load, given that initial range that each node obtains closely relates to the routing load that the node experiences - more precisely, it is to achieve *data balance*[Ganeshan04], by which data are roughly distributed across the partitions evenly.

4.4.3.2 Topology-aware Heuristic Algorithm

The topology-aware overlay construction algorithm in OverMon is very similar to Mercury's overlay construction algorithm, except that a heuristic algorithm is designed to let a new node *choose*, rather than randomly pick, an existing node to join.

Briefly, when a node joins the system, it contacts an existing node. Besides hub configurations and all known neighbours (i.e. processors, successors, and long-distance neighbours), the existing node also sends the network topology that it has snooped, to the new node[12]; this heuristic algorithm is then called by the new node to choose a node that makes the best compromise between network distance and range balance. In more details, the pseudo code of this heuristic algorithm is shown in Figure 4-9, and explained as follows.

Firstly, (referring to pseudo code line 1-3), once the topology and the known neighbour list is received by a new node, the shortest paths (SPs) rooted from itself to each destination within the list is calculated. This SP set is further transformed into a map indexed by the distance (i.e. the length of SPs in hops), such that the destinations that are of the same length of SP are grouped in the same level, and the levels are sorted in ascending order.

[12] The topology can certainly be snooped by the new node itself, rather than being sent by the existing node, but this will cause unnecessary delay for the new node, given that the network is stable with very high probability each node obtains the same network topology.

Secondly, (referring to pseudo code line 4-11), at each level, only one node that has the largest range is picked and added to a new map; again the map is indexed by the distance and sorted in ascending order. To optimize at this stage, the number of loops is limited to *LoopDepth* (pseudo code line 4), meaning only the top *LoopDepth* levels of nodes (i.e. the *LoopDepth* nearest), are considered. *LoopDepth* is a configurable parameter and by default being set as 5. The larger it is, the less important the factor of distance is, as this means more distant nodes wth larger ranges have more chances to be chosen.

Then, it comes to the loop of choosing the node to join, (referring to pseudo code line 12-32). Starting from choosing the first node (i.e. the nearest node) as the *joinPeer*, and comparing it with the rest of nodes: if the *comparePeer*'s range is too small to be split, the loop continues to process the next node in the map (line 17-19); otherwise if the *joinPeer*'s range is too small, it is given up and replaced by the *comparePeer*, and the loops is continued and re-started (line 20-24).

If none of the *joinPeer* and the *comparePeer* has the minimum range, their differences in range (i.e. *range_diff*) and in distance (distance_diff) are calculated (line 25, 26). If the ratio of *range_diff* to distance_diff is larger than *BSFitness*, i.e. the gain in range is larger enough than the gain in distance, the *comparePeer* is chosen as the *joinPeer*, and a new loop starts until the rest of nodes in the map is checked in a similar way (line 27-32). Here, *BSFitness* is also a configurable parameter, by default being set to 1.0. The larger it is, the less important the factor of range is, as it is more critical on the gain in range.

By using this heuristic, the network level overhead is expected to be reduced, the balance between distance and range is expected to be achieved, and at the overlay level, the initial range assigned to nodes is expected to be balanced. The evaluation of this algorithm is presented in Chapter 6.

```
   topologyAwareJoin( localNode,hubName,    topology,knownPeers_all )
(1)    knownPeers_hub = knowPeers_all(hubName)
(2)    shortestPathSet = caculateSP(topology,localNode,knownPeers_hub)
(3)    shortestPathSet-->distancePeerMap // group&sorted by SP length
(4)    distCount = distancePeerMap.size()
(5)    roundMax = (distCount >LoopDepth)? LoopDepth: distCount //default is 5
(6)    peerRangeDistMap = {}
(7)    for (each level in roundMax)
(8)    {
(9)        peerAtLevel=choosePeerForMaxtRange(distancePeerMap[level])
(10)       peerRangeDistMap.add(peerAtLevel)
(11)   }
(12)   joinPeer = peerRangeDistMap.first()
(13)   for (rest peers from peerRangeDistMap)
(14)   {
(15)       comparePeer = peerRangeDistMap.next()
(16)       if(comparePeer.range < min_splitable_range)
(17)       {    // range is too small   , not good to join
(18)            continue
(19)       }
(20)       else if (joinPeer.range == min_range)
(21)       { // range is too small, not good for split in future
(22)            joinPeer=comparePeer
(23)            continue
(24)       }
(25)       range_diff = comparePeer.range - joinPeer.range
(26)       distance_diff = comparePeer.distance - joinPeer.distance
(27)       fitness = ratio of range_diff / distance_diff
(28)            if (fitness > BSFitness)    // default is 1.0
(29)       {
(30)            joinPeer = comparePeer
(31)       }
(32)   }
```

Figure 4-9: The pseudo code for topology-aware join algorithm

4.5 Possible Design Alternatives

This section discusses some of the possible design alternatives.

Firstly, it might be possible to integrate the control overlay and the data overlay into one overlay network. For example, the overlay might be constructed using a tree-based structure, given that the nature of application level multicast in the control plane is to construct a distributed tree, and tree structures to support range queries have been studied [Zheng06, Schmidt03]. Furthermore, given that the nature of aggregation is a bottom-up tree convergence, adopting a tree structure might be able to support all of multicast, range query, and aggregation. However, questions that remain to be asked are: to support multi-attribute range query, would multiple tree structures need to be constructed and maintained simultaneously? If so, for a multi-attribute range query, would complexity increase exponentially if the approach of Space Filling Curve is adopted to resolve the query [Lawder00]?

Secondly, a centralized data repository could be introduced as an economical solution, given that out-of-band resources are much cheaper than the limited resources at the edge routers, and the size of networks that OverMon supports is not extremely large. However, to eliminate the single point of failure implicit in such an approach, and to provide the flexibility of retrieving data from any edge router, this centralized repository might only be a complementary means. For example, by defining a time limit that measurement results are stored in the distributed repository, out-of-date measurement results could be transferred to a centralized data repository, and specialized mechanisms, e.g. integrating with DBMS, could be deployed to provide more advanced functionalities of data management.

OverMon is designed to use separate overlay networks to support multicast and range query (without aggregation), and store data in a distributed fashion across the participating routers; these possible alternative designs are interesting topics for future study, particularly the centralized data storage might be helpful to make OverMon practically more deployable.

4.6 Summary

This chapter has presented the design details of OverMon.

It firstly outlines OverMon from a high level, introducing the principles of design and the

system infrastructure. In a nutshell, OverMon is to apply the cutting edge technology of overlay network to build a distributed network monitoring system. More importantly, by exploiting internal gateway protocols, such as OSPF, overlay networks can be constructed with topology-awareness, which not only saves the overhead caused by normal overlay maintenance, but also improves the overlay's performance by minimizing its impact on the underlying network.

Then, major components are presented in details, namely topology tracker, control overlay, and data overlay.

In Section 4.2, the topology tracker is firstly introduced: it reconstructs topology information by passively capturing OSPF LSU packets and parsing them with relevant LSA information. This topology information is then utilized by OverMon in building control overlay and data overlay networks.

In Section 4.3, the application level multicast tree is constructed for the control overlay, by which control information, i.e. the specification of active measurement task is disseminated to edge routers that are required to participate the task. With network topology being available, to build multicast trees with high quality, there is no need for OverMon nodes to probe each other in order to obtain approximate topology information, thus can greatly reduce bandwidth overhead, and at the same time improve the performance of the multicast tree.

In Section 4.4, data overlay is presented, which is a virtual repository constructed on edge router's memory to index measurement results, herein to support range query on measurement results. For this purpose, the distributed data structure and corresponding distributed algorithm is an application of Mercury [Bharambe04]. However, a heuristic algorithm is designed to make the decision of choosing a node to join become more configurable and reasonable, towards minimizing overlay's impact on underplaying network layer via topology-aware overlay construction, as well as to mitigating the imbalance in data distribution.

Lastly, possible design alternatives are discussed in Section 4.5, for example, to integrate the control overlay and the data overlay into one overlay network, or to introduce a centralized data repository as a complementary means to the distributed repository.

With the design of OverMon discussed so far, such an overlay based monitoring system is promising to achieve good performance in terms of being flexible to address dynamically changing topologies, scalable to address increasing network size, and extensible to address high levels of complexity and heterogeneity. The design is implemented as a Java-based

prototype as described in Chapter 5, and is evaluated by both simulation and emulation as described in Chapter 6.

Chapter 5

Implementation

Following OverMon's design in the previous chapter, this chapter discusses OverMon from a software engineering perspective, including implementation details of the prototype that was developed. It firstly provides a high level view of the software infrastructure in Section 5.1, and the API functions and data structures in Section 5.2. Then the control overlay and data overlay are discussed in Section 5.3 and 5.4, respectively, particularly with their inter-node operations, intra-node software structures, and the message exchange mechanisms. Finally, Section 5.5 summarises the contents of this chapter.

5.1 Software Infrastructure

The prototype of OverMon is implemented in Java[13]; it has been built with JDK1.5 under Eclipse 3.1.0 but is portable to different compilers and platforms. It is a three layered system with heavy use of object oriented facilities, thus it is easy to extend to support new features. Figure 5-1 outlines the division and interaction between the different layers, showing the major modules and interface methods of the software structure.

The top layer, the RMI (Remote Method Invocation) layer, consists of one major module named as OverMonRMIServer. This layer accepts *monitoring tasks* issued by the users (i.e. act as the RMI clients); a monitoring task can register a new measurement task, or retrieve a measurement result. It thus interfaces with the supporting overlay layer via the *rmiTaskInit()* method; once the monitoring task has been completed by the supporting

[13] For the data overlay in OverMon, although the C++ code of Mercury is claimed to be publicly available, experience attempting to use it and correspondence with the author indicated that it had not been fully tested in a real network environment. Therefore, OverMon has its own implementation in Java, following the documented design of Mercury.

overlay layer, the result is returned to users via *callBack()*, a call-back style interface method.

The supporting overlay layer has three major modules, namely *CtrOverlayMgr*, *TopologrMgr*, and *DataOverlayMgr*. As their names indicate, the *CtrOverlayMgr* module mainly deals with the functionalities of the control overlay, including construction of ALM trees (via the *alm()* method) to disseminate the measurement task configurations to the appropriate participating nodes, and to schedule the corresponding active measurement (via the *schedule()* method) according to the task configurations. Similarly, the *DataOverlyMgr* module deals with the functionalities of the data overlay, including construction of the hub-based overlay network (via the *hub()* method), and routing data overlay messages (via the *route()* method). Both *CtrOverlayMgr* and *DatarOverlayMg* need to interface with the network layer, to set up network connections with other overlay nodes; in addition, both of them obtain topology information provided by the *TopologyMgr* module (via the *topology()* method), which interfaces with the *OSPFSnooper* module at the network layer using an instance of the *observer* pattern – i.e. a publish/subscribe relationship exists between *OSPFSnooper* and *TopologyMgr*, thus *TopologyMgr* is notified by *OSPFSnooper* when there are any changes in the topology graph.

The network layer performs network level functionalities, including conducting active measurement tasks by sending UDP based probe packets (i.e. *UDP_AM* in Figure 5-1); supporting the supporting overlay layer by making TCP/UDP connections to other nodes (i.e. *UDP_Worker* in Figure 5-1 for the control overlay, and *TCP_Worker* in Figure 5-1 for the data overlay); and providing topology information by passively capturing OSPF packets and continuously maintaining a topology database. In addition, when an active measurement finishes, the *UDP_AM* module interfaces with the overlay layer to pass the measurement results to the *DataOverlayMgr* module, from which the results can be routed to a node whose range covers the values of the measurement results.

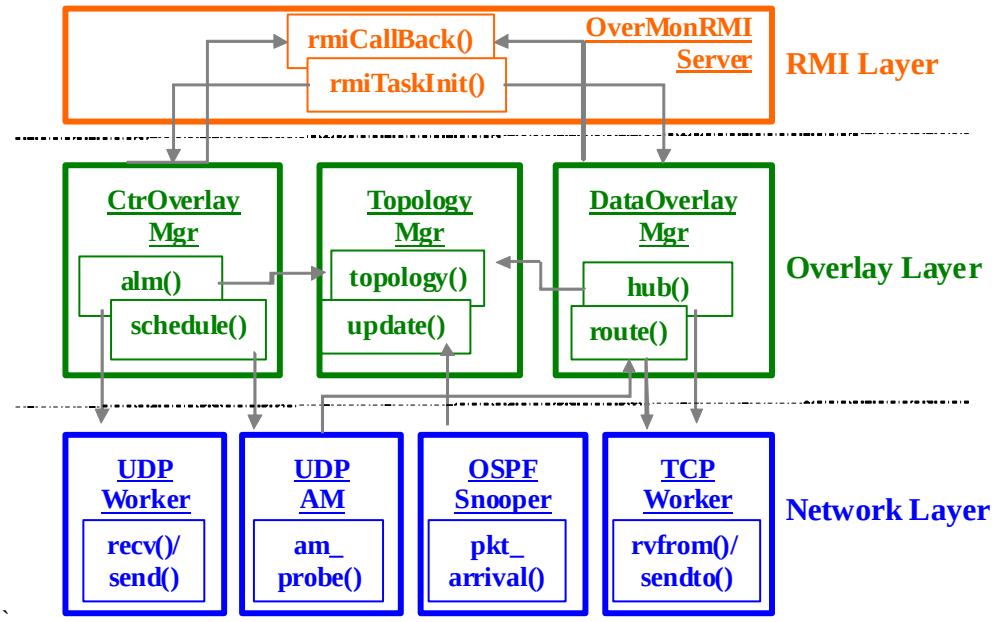


Figure 5-1: The 3-layered software infrastructure with major interfaces

5.2 User Level Interfaces

User level interfaces refer to setting up the interfaces that allow OverMon users to perform monitoring tasks. The design is driven by the desire to make OverMon a light-weight monitoring system, but flexible and extensible in terms of providing a large configuration space to users. The topics discussed in this section include the API functions that are provided to users, and the utilization of Java RMI, which not only facilitates these interactions, but also provides an efficient means to detect and control the run-time status of an OverMon node for experimental purposes.

5.2.1 API Functions

Basically, the API functions provide to users of the ability to perform a monitoring task, which include initiating an active measurement task with specified parameters, cancelling a registered measurement task, querying a measurement result, and obtaining a measurement result reported by OverMon, as shown in Table 5-1.

The active measurement that OverMon supports is based on the standard measurement metrics defined by the IETF IPPM (IP Performance Measurement) working group, as

introduced in Section 2.2.2.2. Given that throughput metrics are not specified by IPPM, OverMon supports throughput measurements based on technologies such as VPS (Variable Packet Size), PPTD (Packet Pair/Train Dispersion), and SLoPS (Self-loading Periodic Streams), which in common need to send probes into the network for measurement purposes. To accommodate all these standards and techniques, the correspondingly high level data structures are defined in Java classes, as shown in Appendix A.

Table 5.1: The API functions of OverMon

```
// to initiate an active measurement
public MonitoringTaskResult registry (IPPMMetric, ProbeStructure, TimeToLive, ReportFlag)

//to cancel an active measurement
public boolean cancel (taskID)

//to query measurement results satisfying the given range
public IPPMeasurementResult[] query (AttributeName, ValueRange)

//obtaining an active measurement result that is reported by OverMon
public IPPMeasurementResult[] report (taskID);
```

Specifically, to register a new measurement task, four parameters need to be provided: *IPPMMetric*, *ProbeStructure*, *TimeToLive*, and *ReportFlag*. Among thme, *IPPMMetric* specifies the performance metric that to be measured and the methodology that is to be used; *ProbeStructure* specifies all the information that is needed to perform the active measurement; *TimeToLive* specifies a default time-out period for measurement result to be stored in OverMon; and *ReportFlag* specifies whether the measurement result should be actively reported by default, i.e. be pushed to users when the measurement task finishes.

Once an active measurement task is successfully registered with OverMon, the *MonitoringTaskResult* is returned to users, which contains a unique *taskID* for the registered measurement task. With this ID, the task can be cancelled by users before it is executed. To guarantee the uniqueness of each taskID within the whole system, the taskID is a combination of a numeric descriptor uniquely generated by the initiating node, and the unique ID of the task initiating node, e.g. the IP address.

Users can issue range queries for measurement results with two parameters: *AttributeName* and *ValueRange*. *AttributeName* specifies the particular attribute that is of interest, and can be any single attribute from the set {*TaskID, TimeStamp, Throughput,*

Packet_Loss, Packet_Delay, Jitter}, and *ValueRange* specifies the upper and the lower bound of a range that is to be queried.

Once the query is routed along the data overlay, each node whose range satisfies the querying condition sends its result to the querying initiating node, where the results are aggregated into a list and returned to users. In the case when measurement results are pushed to users, once the task finishes, each probe-receiving node sends its measurement results directly to the task initiating node; there they are aggregated into a list and returned to users.

5.2.2 Utilization of Java RMI

Java Remote Method Invocation (Java RMI) is a well-known infrastructure that enables the invocation of methods of remote Java objects at runtime from other Java virtual machines (JVM) that might be on different hosts. Basically, an RMI interaction involves a *client*, a *server* and a *registry*. To make a call to a remote object, the client first looks in the registry for the object on which it wishes to invoke a method. If the object exists, the registry returns a reference to the object, which is registered by the server and can be used by the client to invoke any methods implemented by the remote object.

Java RMI only provides a framework to invoke a remote method; the actual usage of this framework is left to users. In other words, as long as the server and the client extend the *java.rmi.Remote* interface, any user-defined objects and methods can be implemented.

In OverMon, an OverMon node acts as the server, and a user of OverMon acts as the client. The interfaces for the server and the client, as well as the RMI task that can be passed from the client to the server, are defined in Table 5-2.

To perform a monitoring task, the RMI interaction procedes as follows:

1. When an OverMon node starts, an instance of the *OverMonRMIServer* is generated; this instance rebinds the methods declared in the *OverMonRMIServer* interface in the registry with a name, in the form"*//IPAddess/Port/ClassName*".
2. When a user, i.e. the RMI client, tries to perform a monitoring task, the name of the API call and the associated configuration parameters are instantiated as *operation* and *parameters*, the two member variables encapsulated in an *OverMonRMITask* instance.
3. Then, the RMI client looks for the name of the remote method in the registry. If the

lookup is successful, a reference to the remote object is returned by the registry. By using this reference, a remote method, such as *registerNoticableClient()*, is called to register the client as noticeable on the server, and *rmiTaskInit()* is called to pass on the monitoring task from the RMI server to the supporting overlay layer to be processed.

Once the monitoring task is complete, the supporting overlay layer first returns the result to the *OverMonRMIServer* instance. The *OverMonRMIServer* instance notifies the user by calling the *rmiCallBack()* method, which in turn calls the client's *notify()* method, enabling the result of the monitoring task to be returned to user.

Table 5.2: The interfaces definitions of Java RMI

```
// interface definition for RMI server
public interface OverMonRMIServer extends Remote {

        public void rmiTaskInit(OverMonRMITask task)
                                        throws RemoteException;

        public void registerNoticableClient(OverMonRMITask rmiTask,
                                                OverMonNotifiableClient rmiClient)
                                            throws RemoteException;

        //public void rmiCallBack(OverMonRMITask task, MonitoringTaskResult result)
        // this method is not to be invoked by the RMI client,
        // rather, it is called by the server to notify the registered client by calling the notify()
          method provided by the RMI client
}
```

```
// interface definition for RMI client
public interface OverMonNotifiableClient extends Remote {

        public void notify(OverMonRMITask rmi_task, MonitoringTaskResult rmi_result)
                        throws RemoteException;
}
```

```
// interface definition for RMI task
public interface OverMonRMITask {

        public byte getParameters();
        public byte getOperation ();
        public boolean equals(Object o);
}
```

The benefit of utilizing Java RMI in OverMon can be summarised as follows. Firstly, as a complete distributed solution, from one control console, a user can choose any OverMon

node (i.e. each is an RMI server) to perform a monitoring task. Secondly, it releases a user's thread from waiting, i.e. the results of registering a new measurement task or retrieving a measurement result can be sent to users using an asynchronous call back. Thirdly, it becomes an efficient means to detect and control the run-time status of an OverMon node through the extended API functions that are specially designed for the experiment purpose.

5.3 The Control Overlay

This section introduces the control overlay of OverMon, regarding its inter-node operations, intra-node software structures, and the message exchange mechanism.

5.3.1 Inter-node Operations

In OverMon, the control overlay is coordinated mainly by *CtrOverlayMgr*, the Java class acting as the control overlay manager. Its major functionalities include constructing the ALM tree to disseminate the configuration parameters of a measurement task to the task participating nodes[14], and to schedule the measurement task according to the task configuration.

Recall that the ALM routing algorithm in OverMon is simple in that the membership of a multicast session is statically pre-defined, and the duration of a multicast session is short enough such that any maintenance or optimization of the tree is not required. To construct the multicast overlay, the Steiner tree is only calculated by the root node based on the snooped network topology; all other nodes just relay and respond to the overlay construction messages by following the generated tree. As a result, OverMon nodes play different roles in such a multicast session, as described in Section 4.3.3. Accordingly, inter-node operations for each role, following the sequence of their occurrence, are listed in the Appendix B, Table B-1.

5.3.2 Intra-node Structures

Although the construction of multicast tree for a single multicast session does not seem complicated, given that multiple sessions may co-exist on one OverMon node, and the

[14] Cancelling a registered task is processed in a similar way.

role it plays may be different from session to session, efficient data structures and software design are required to cope with this situation. In Figure 5-2, the major intra-node operations and interfaces of CtrOverlayMgr with other components are illustrated.

Basically, for an OverMon node to accommodate multiple sessions at the same time, each session is described by an *ALMSessionInfo* data structure (i.e. *sessionInfo* in Figure 5-2), which not only encapsulates the original configuration information of the measurement task (passed from the RMI layer), but also encapsulates information regarding the multicast session, i.e. the ID of the root node (rootID) and the sequence number of the session (sessionSeq). The sessionSeq is a contiguous sequence number generated by each node when it initiates a new multicast session; by combining sessionSeq and rootID, a multicast session can therefore be uniquely identified. The activity of each multicast session is further wrapped into an *ALMSession* (i.e. *session* in Figure 5-2), thus the Steiner tree calculation, as well as the progress of each multicast session, can be guaranteed to be independent from each other.

For each OverMon node, depending upon the role that it assumes, all of the multicast sessions in which it participates are organized into three tables. The entries in these tables are aggressively managed using timers. At every tick, e.g. every other minute, the session at the head of each table is examined to see whether it should be timed out. If the table is for sessions in which the node acts as a leaf node in the multicast tree (i.e. the *Leaf Table* in Figure 5-2), timing out means that the time scheduled to perform the measurement task has arrived. Otherwise, for those sessions in which the node acts as the root node or an internal node, timing out means that the multicast packets have been sent out, however not all the ACK packets have been successfully received. If a node receives all the ACKs that it is waiting for, it deletes the session from the corresponding table, and either acknowledges the upstream parent if it is an internal node, or using call-back to notify the RMI clients if it is the root node in the session.

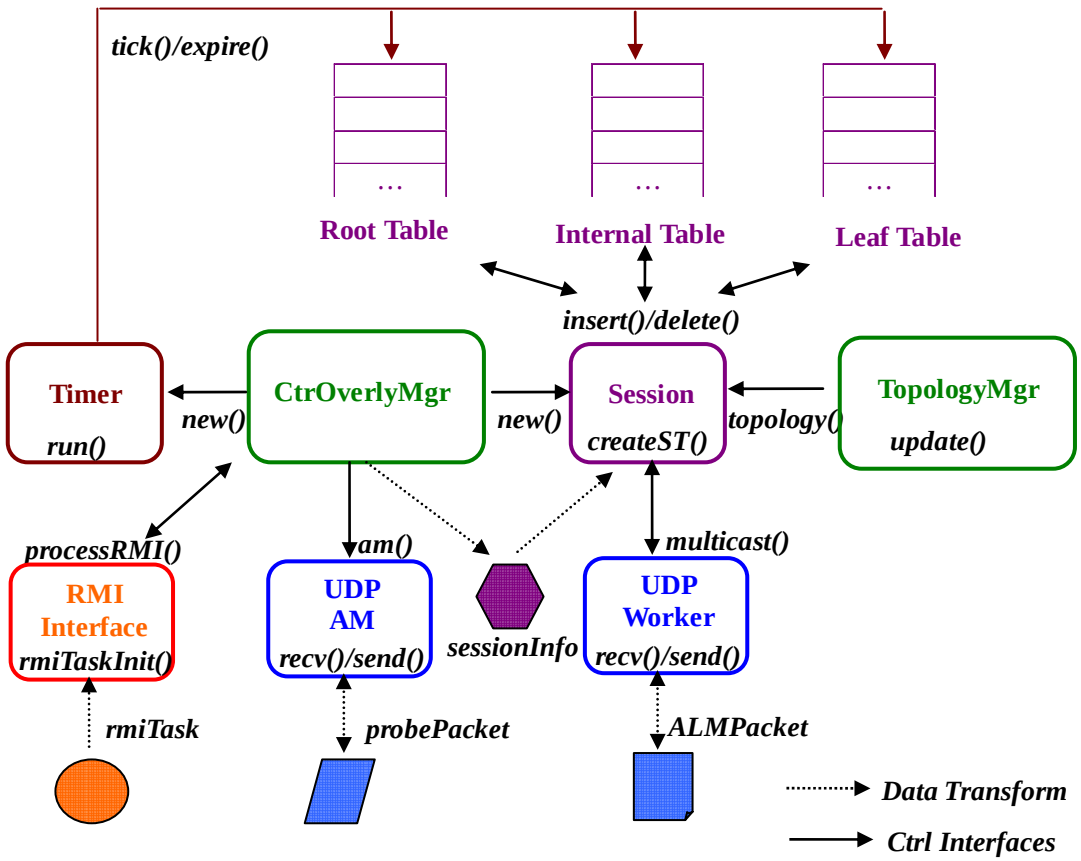


Figure 5-2: Intra-node operations and interfaces in OverMon's control overlay

5.3.3 Message Handling

As seen from Figure 5-2, UDP is the transport protocol used by the control overlay, primarily due to its low latency cost and state overhead relative to TCP. The multicast content is encapsulated into a packet using the *ALMPacket* data structure, which is also used to carry the acknowledgement from a node to its upstream parent node. An ALMPacket is composed of a header part and a data part. The packet header format is shown in Figure 5-3.

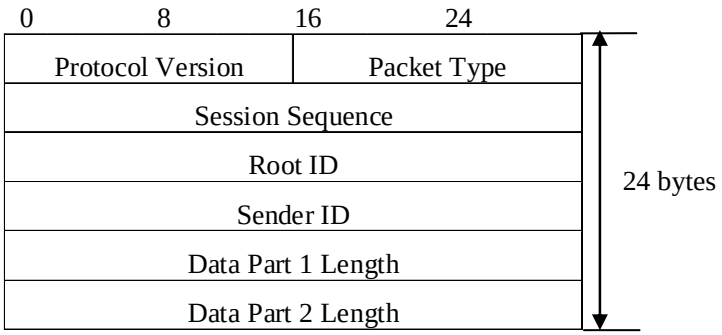


Figure 5-3: The header format for ALM packet

The definition of packet header is rather straight forward: the *Packet Type* field indicates whether the packet is a multicast packet or a multicast_ack packet; the *Session Sequence* field is the contiguous sequence number generated by the root node, whose ID is shown in the *Root ID* field; and the *Sender ID* field contains the ID of the sender of this packet. Lastly, the *Data Part 1 Length* and *Data Part 2 Length* fields specify the lengths of the two segments of the data part, as explained below.

In a multicast packet, the data part consists of two segments: the measurement task's configuration parameters; and the multicast tree map calculated by the root node. Thus, their lengths in bytes are specified by the fields of *Data Part 1 Length* and *Data Part 2 Length* respectively. In a multicast_ack packet, the data part consists of the acknowledging node's answer (i.e. accepting or rejecting the measurement task), as well as a list of downstream children who reject the measurement task. Thus, the *Data Part 1 Length* field specifies the length of the local node's answer, and the *Data Part 2 Length* field specifies the length of rejecting node list (it is zero if none of the downstream children rejects the measurement task.)

5.4 The Data Overlay

This section introduces the data overlay of OverMon, focusing on its inter-node operations, intra-node software structures, and the message exchange mechanism.

5.4.1 Inter-node Operations

The data overlay is coordinated mainly by DataOverlayMgr, the Java class acting as the data overlay manager; its major functionalities include constructing and maintaining the

hub-based overlay network, and routing data overlay messages.

Recall that in the data overlay, nodes are logically arranged into ring-based hubs with each responsible for a contiguous range of values for the attribute that the hub represents; data items and queries can therefore be routed along the hubs and processed by all nodes that might potentially have matching values. To do so, each node maintains three types of neighbouring relationship within a hub, namely successors, predecessors, and long-distance neighbours; as additionally, each node maintains cross-hub neighbours that are in other hubs to support multiple attribute routing. Periodically, random sampling is performed thus system level estimates can be made to improve the long-distance link construction.

Accordingly, the inter-node operations can be grouped into three categories, namely, overlay construction, random sampling, and overlay routing. Given that the overlay construction is complicated in terms of one action (e.g. joining/leaving/long-neighbour establishment) normally having multiple nodes involved, the operations taken by different nodes are listed in Appendix B, Table B-2 ~ B-4. The corresponding message exchange, including the status change of the involved nodes, is illustrated through examples in Figure 5-4. For random sampling and overlay routing, the peering relationship between nodes is simple; the associated operations are listed in Appendix B, Table B-5 and Table B-6 respectively.

As mentioned in Section 4.4.4.2, the overlay construction in Mercury involves a periodic operation of pinging the successor list, i.e. a short list of contiguous nodes further clockwise along the ring other than its successor. This operation can be used to maintain the liveness information of nodes in the system, as a node might depart the system without notifying others; clearly, it introduces considerable network overhead. In OverMon, this is not adopted for the following reasons: since OverMon is deployed on edge routers, the liveness information of nodes is rather stable and the set of edge routers do not change frequently; if there is a departure of an edge router, the *leaveNotifyMsg* message will be sent out actively to notify other nodes; therefore, the correctness of the hub based overlay can be guaranteed. In the case that an unexpected crash happens to an edge router, snooping of OSPF packets will alert OverMon to its disappearance, and OverMon will remove the node from the list of edge routers, which is a configuration parameter initialized when the system is started.

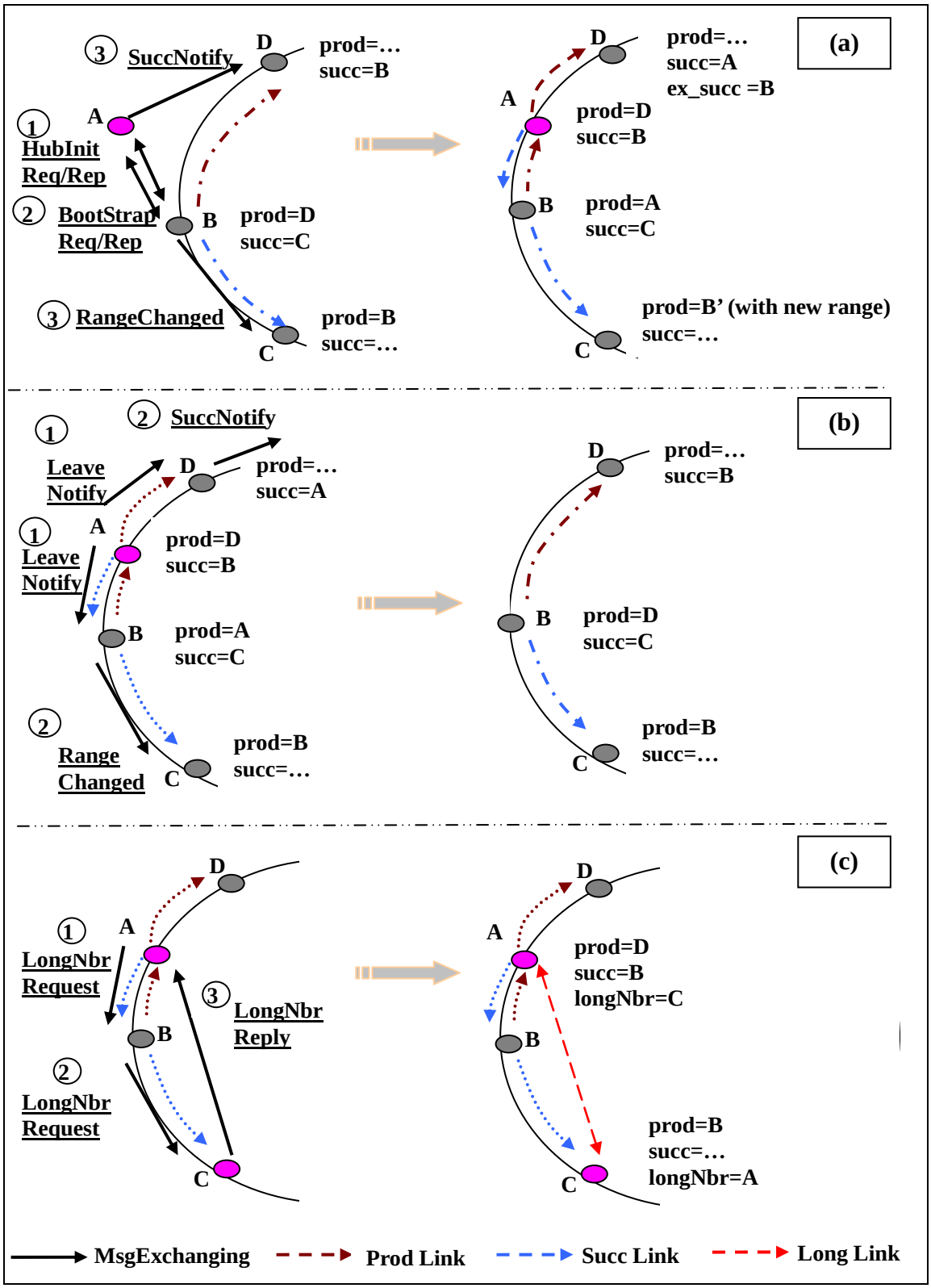


Figure 5-4: Operations and message exchange for data overlay construction, including node's joining (a), leaving (b), and long-link construction(c), with cross-hub pointers are omitted

5.4.2 Intra-node Structures

In Figure 5-5, the major intra-node operations and interfaces of *DataOverlayMgr* with other components are illustrated. As seen, *DataOverlayMgr* interfaces with the *UDP_AM* module once an active measurement produces the measurement result; it also interfaces with the RMI module when a range query is being issued. In both cases, a *routingItem* is generated to wrap the measurement result or the range query, as a routable item to be inserted into, or retrieved from the system (respectively by the components of *Insertion* and *Retrieval*). If the routing item is a range query and the result is retrieved from the system, the *DataOverlayMgr* interfaces with the RMI module again to return the queried result to users; and the *Insertion* component stores the measurement results in a Java vector.

In addition, to set up connections with other nodes, the *DataOverlayMgr* interfaces with the network layer module *TCP_Worker* through the *MsgSwitch* component. Essentially, the *TCP_Worker* module provides physical network connections with other nodes by using TCP as the transport protocol, while the *MsgSwitch* component is to provide and process three types of interface methods to other components that need to interact with other nodes, e.g. *Sampling* and *Construct* as part of *HubMgr*, and *Retrieval* and *Insertion* as routing facilities. In more details, by calling the *routing()* method, a message is sent to the next hop that is calculated by using the routing algorithm as discussed in Section 4.4; alternatively, by calling the *sending()* method, a message is sent directly to a node without being routed; and by calling the *delivery()* method, messages are dispatched to different components according to the message type. In the case that a routing item is for the local node, it is immediately delivered to the responsible component without being passed onto the network. Clearly, the *Retrieval* and *Insertion* components call the *routing()* method most, while the *Sampling* and *Construct* components call the *sending()* method most.

The calculation of next hop in the *MsgSwitch* component is based upon the information provided by the *HubMgr* component, which maintains a set of hubs and each hub performs independent sampling and constructing. In other words, when a node is a member of multiple hubs, the sampling and constructing operations are processed multiple times, once for each particular hub.

Lastly, in topology-aware overlay construction, the *TopologyMgr* module provides topology information to the *Construct* component, which calculates the fitness of choosing a bootstrap node when a node joins the system. A timer is set up for periodic

operations, such as randomly sampling, long-distance link repair, and histogram construction.

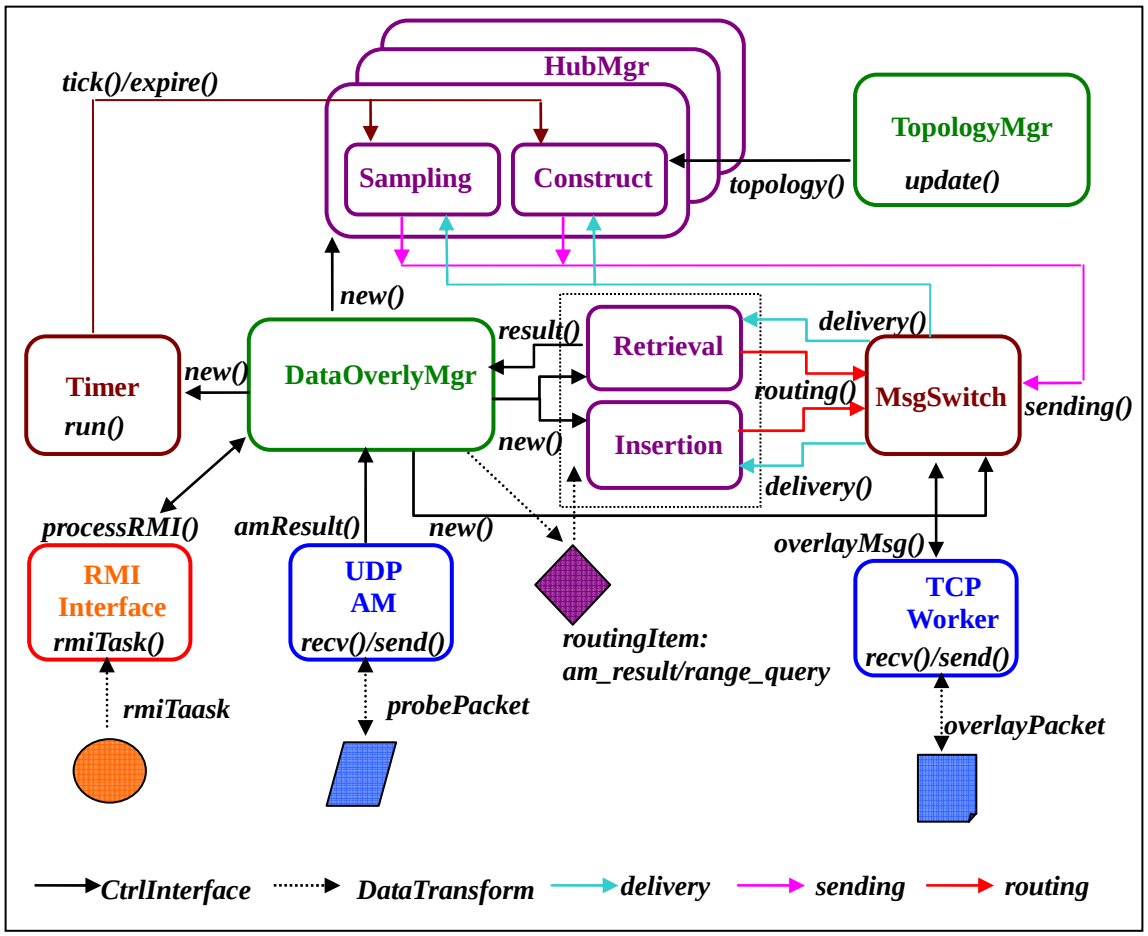


Figure 5-5: Intra-node operations and interfaces for OverMon's data overlay

5.4.3 Message Handling

For the data overlay, many more different message types are used than in the control overlay; the Java based XML (Extensible Markup Language) messaging mechanism is therefore employed, which provides standard SAX (Simple API for XML) API to parse a message that is encoded in XML format.

In short, the SAX parser implements an event-driven interface, and events are invoked when XML's special features, such as text, elements, processing instructions, comments etc, are encountered during the parsing process; the invoked events are further processed by calling *callback* methods that are defined by the user.

In OverMon, the Java class defined for OverMon messages extend the *org.xml.sax.helpers.DefaultHandler* class and overwrite the *startElement*() method, which will be invoked by the SAX parser at the beginning of every element in the XML messages. Therefore each member variable correspondingly defined in the message class can be retrieved by taking specific actions. Note that this message processing mechanism takes place at the overlay layer; at the network layer, an XML message is transformed into bytes by serialization when it is passed onto a socket and is reassembled once it is received from a socket.

Similar to the control overlay, the messages that are exchanged for the data overlay are composed of two parts: the message header and the message body. In the message header, basic information about a message is specified, such as message ID, message length, message type, the ID of the sending node, as well as the sending node's IP address and port number. While in the message body part, different messages have different definitions and different contents. In Table 5-3, a *BootStrapReply* message is displayed as an example; it shows that except for the header, the information contained in the *BootStrapReply* message includes the range assigned to the joining node, the range left for itself, and its predecessor with which the joining node needs to contact.

Table 5.3: An example of data overlay message in XML format

```
<OverMonDataOverlayMsg Version="1.0" ID="D7B5…8854" Content-Length="143">
        <Header Type="3" Hops="1">
                <Host ID="" BFCA...FF242 " IP="192.168.16.2" Port="1805"/>
        </Header>
        <BootStrapReply HubName="TaskID" AssignedRange="320-640" LeftRange="0-320"
EXProcd="ip=192.168.59.1 ,port=1805 ,range[0-640]">
        </BootStrapReply>
 </OverMonDataOverlayMsg >
```

By using this XML-based messaging, the inherent flexibility and extensibility of XML can be leveraged, which enables new operations and services to be added easily and quickly. Also, the readability of this messaging mechanism mitigates the difficulty in developing and debugging a distributed system like OverMon at the experimental stage.

5.5 Summary

This chapter has presented the implementation details of OverMon from a software engineering perspective, particularly focused on the design of internal and external interfaces.

It firstly outlines the software infrastructure in Section 5.1. Then it describes the user level interfaces in Section 5.2. The implementation details of the control overlay and the data overlay networks are presented in Section 5.3 and Section 5.4 respectively.

Using object-oriented methods, OverMon is implemented as a three-layered system, with the RMI layer on the top, the overlay layer in the middle, and the network layer at the bottom. By this careful design of interfaces, it allows a seamless interaction between users and the OverMon software, as well as between different software components within OverMon. Together with extensible XML-based message handling, new functions and modules can easily be added, without impacting existing ones. In addition,

In addition, by using Java RMI, the result of performing a monitoring task can be returned to users in an asynchronous call-back style, which allows users to obtain the results in real time, but without blocking the user's thread in a synchronous call. Furthermore, since the run-time status of nodes can be detected and controlled through the extended RMI interface, it mitigates difficulty in debugging and developing OverMon.

In summary, to implement a fully distributed application-level overlay network, it is non-trivial and notorious for being difficult and tedious [Li04]. The principle of software design in OverMon was to maximize the flexibility and extensibility inherited from overlay techniques, without sacrificing the performance. As the proof of the concept, the prototype is developed following this principle, and shows that it is feasible to build an overlay-based network monitoring system like OverMon. By using this prototype, the performance of OverMon is evaluated in Chapter 6.

Chapter 6

Evaluation

This section focuses on the evaluation of OverMon. In Section 6.1, it starts with a discussion of how to choose an efficient approach to evaluate a distributed network system. Then, the proposed solution is introduced in Section 6.2, and the concrete experimental network setup for evaluation is covered in Section 6.3. Next, the evaluation results of control plane and data plane are presented in Sections 6.4 and 6.5, respectively. Finally, Section 6.6 summarizes the whole chapter.

6.1 Evaluation Strategy

In this section, various evaluation methodologies for large scale distributed network systems are firstly discussed in Section 6.1.1, then the tactics for selecting a suitable approach for OverMon is analysed in Section 6.1.2.

6.1.1 Testbed vs. Simulation vs. Emulation

It is widely accepted that distributed network systems are difficult to evaluate. Generally speaking, the methodologies that can be used for evaluation fall into three categories, namely testbed, simulation and emulation. [15]

A testbed is a test environment dedicated to evaluation. It offers a high level of realism in resembling the characteristics of target production networks. But this realism comes at significant construction cost. More importantly, due to their highly distributed nature,

[15] Modelling sometimes can be counted as a means of evaluation; however, it is taken in this dissertation as an algorithmic approach, and complementary to the three system-level evaluation approaches.

runtime behaviours of large-scale network protocols are not possible to analyze on a small, experimental testbed. Therefore, a testbed is not practical for large-scale network protocols, or complicated distributed network systems with highly varying network conditions. Here, network conditions are referring to higher level concepts, such as a link's bandwidth, capability, a router failure, topology changes etc.

Simulators, such as NS [NSWeb] and OPNET [OpNetWeb], offer an efficient event-driven execution model by requiring the protocol under test be rewritten according to the simulator's event-driven model. There is no actual network traffic in simulators, and the functionalities provided by the supporting modules are merely logical operations; the simulated protocol can not be tested using real implementation code, but must be refined and converted to a real implementation later. The main issue with simulation is that it only provides a synthetic, conceptual, network environment using a virtual timescale. Since the fully controlled simulation environment is decoupled from any external traffic or system, simplified assumptions may result in inaccurate representation of traffic dynamics seen in real-world environments.

Emulators, such as EMPOWER [Zheng04], NIST Net [NISTWeb], Emulab [White02], and MARS [MARSWeb], solve the verification and validation problems by directly executing unmodified real-world code in a network testbed environment. In other words, it can be regarded as real-time simulation that uses real computer systems and networks as the platform; and the protocol modules, i.e. the modules under test, are real implementations interacting with the protocol stack of the underling emulator host[s]. However, one of the major tasks of network emulation is to generate specific network conditions and traffic dynamics as required. Here, traffic dynamics are referring to packet-level concepts, such as packet delay, packet loss etc, whereas network conditions are referring to higher level concepts, such as a link's bandwidth, capability, a router failure, topology changes, etc. To achieve this, typically an emulator software module has to be run within the kernel of the emulator host. Since the context switching time of processes is large, and the maximum number of processes in an OS is limited, the major issue with emulation is its scalability in terms of its overall emulation capacity, and the capability of emulating specific network parameters such as maximum bandwidth and packet delay of the system [Zheng04].

In summary, among these three methodologies, a testbed is expensive; simulation lacks realism; and emulation requires scalability.

6.1.2 Ideal Tactics and Chosen Approach

Since each evaluation methodology has its strength and weakness, an ideal approach is to combine them together to tackle the issues of cost, scalability and realism. Such an approach can be described as follows, although in practice, it might not be possible or necessary to strictly abide by each step.

- Beginning with scenarios of small size, firstly the system can be evaluated by both simulation and emulation, with identical network conditions and run-time parameter configurations.
- By comparing the results from simulation and emulation, the simulation methodology can be verified, and the reliability and limits of the simulation software can be determined.
- Then, scenarios of large networks, or scenarios with varied configuration parameters, can be designed and setup for simulation.
- Once the simulation results show that the system has the desired behaviours, the system can be implemented and deployed on a testbed or larger emulation environment for further evaluation.
- Finally, the system can be deployed into a real environment.

In the case of OverMon, since the motivation is to investigate the feasibility of applying overlay techniques in the domain of network monitoring with concrete usage scenarios and reasonable system assumptions, it is crucial to evaluate the system under a practical network environment where the prototype can be fully tested, and the performance under real network conditions can be accurately predicted. Therefore, an emulation approach is firstly considered.

However, most emulator tools are very hardware-specific; they normally require a dedicated environment constructed for a specific purpose. Furthermore, when considering the issue of scalability in emulation, i.e. the limited size of networks that can be emulated, simulation also needs to be considered.

OverMon is a distributed network system where an application level multicast tree and a data overlay network is built upon the network layer; although the underlying network layer supports the overlay network, and its impact cannot be ignored, performance metrics such as number of hops, link stress and stretch are of primary importance. This is in contrast to the fact that most traditional network simulation tools, e.g. NS-2, focus more at network level metrics such as link throughput, packet delay and packet loss. A number of P2P simulation systems have appeared targeting the overlay layer, but they are either at their early stages of development [PeerSimWeb] or particularly designed for

DHT-based overlay networks [P2psimWeb].

Therefore, the evaluation of OverMon should be based on a combination of emulation and simulation. Unfortunately, existing tools for emulation and simulation *cannot* be straightforwardly used. To tackle this issue, an emulation solution is firstly proposed in Section 6.2, which can be utilized to evaluate both control plane and data plane. As to simulation, OverMon's control plane and data plane are treated differently: a customised simulation solution catering for OverMon's control plane is discussed in Section 6.4.2.2; for the data plane, since the building block of Mercury has been extensively simulated by the original authors, its evaluation in OverMon is through emulation only.

Note that, for the time being, both emulation and simulation in OverMon are not taking the timing of overlay construction and maintenance, as well as the dynamic changes in network conditions, into account, due to the fact that the activities of initiating a monitoring task and retrieving a measurement result is not critically delay-sensitive, and an overlay network formed on edge routers is quite stable.

6.2 Building an Emulation Environment

This section presents an emulation environment constructed for the evaluation of OverMon. It firstly shows that by leveraging the virtualization technology of Xen, multiple virtual machines can be created on one physical machine, and these virtual machines can be configured for emulation purpose. Then it presents an emulation toolkit built upon Xen; by using this toolkit, emulation networks are setup automatically, and OverMon can be efficiently deployed on the emulated networks.

6.2.1 Virtualization Technology and Xen

Virtualization technology allows users to simultaneously run several guest operating systems on top of the virtualisation layer on a single computer system. With hardware becoming cheaper and more capable, virtualization technologies are increasingly feasible and important. It is widely accepted that virtualization potentially brings the benefit of maximizing system utilization while reducing server counts and support costs.

Xen is a virtualization technology originated from the University of Cambridge [Barham03]. It enables a single machine to run multiple independent guest operating systems concurrently in separate virtual machines (VMs). These VMs, also termed as

guest domains, are completely isolated from any of the others running on the same machine, which provides the illusion of an isolated physical system for each of the guest operating systems. By isolating VMs, fault tolerance can be preserved between virtual machines. For example, if one guest operating system crashes, it will not take down the whole machine, just its own virtual machine.

The architecture of Xen, as shown in Figure 6-1, is layered, and the lowest and most privileged layer is the Xen Virtual Machine Manager (VMM). For network connections, each domain network interface (vif) is connected to a virtual network interface in dom0 by a point to point link which can be effectively viewed as a “virtual crossover cable”. Domain 0 takes control of each vif’s access to the host machine’s physical network devices, and “*switches*” each packet seen at the host’s physical network device to the appropriate vif. Consequently, each *vif* in a guest domain appears as a normal network interface card (NIC) to its own domain. In order to “switch” incoming packets through the correct *vifs* to a guest domain, and outgoing packets from *vifs* to the correct physical device, domain 0 is responsible for performing proper ARP configuration for each active guest domain to generate the desired IP routing path inside the local Xen machine and across multiple Xen machines; then it handles the packets by using standard Linux network utilities, such as bridging, routing, NAT, etc.

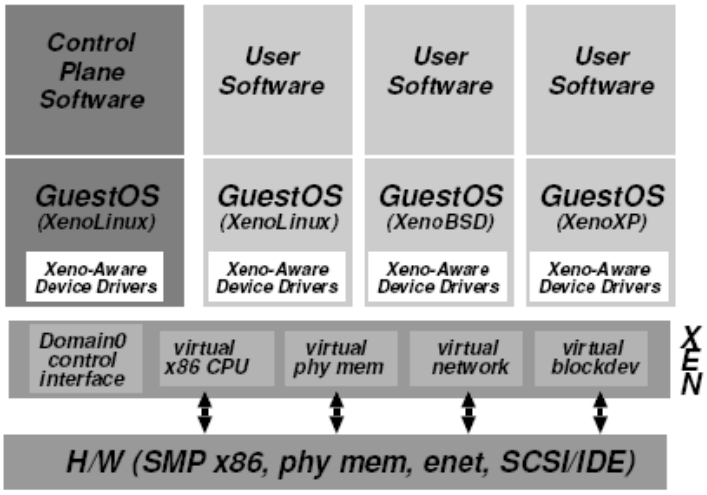


Figure 6-1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including Domain0 running control software in a XenoLinux environment [Barham03]

Figure 6-2 shows a simple example of the network setup on two Xen machines. The two machines are connected by a Gigabit Ethernet switch and on each of them, two guest domains are created. In each guest domain, a different number of virtual NICs are configured and bridging is used to connect the virtual interfaces to which these NICs are attached.

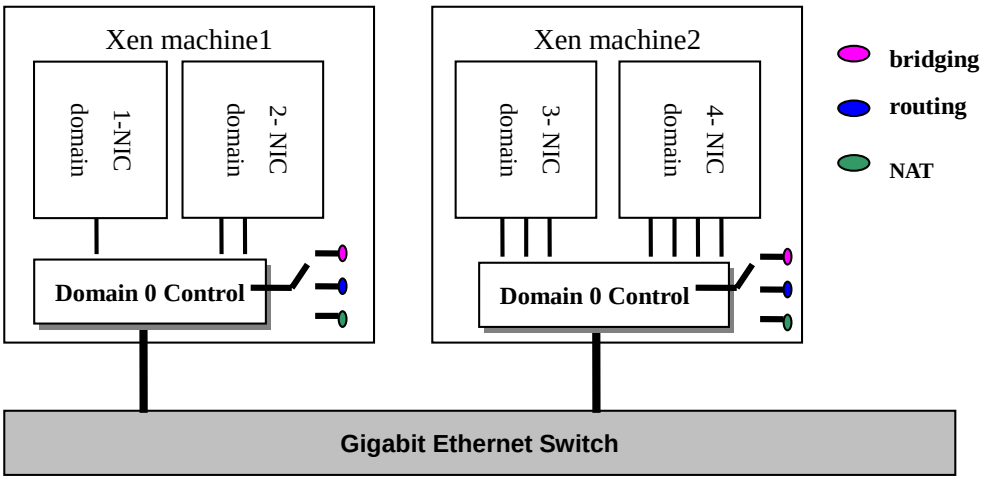


Figure 6-2: An example of virtual nodes on two machines running Xen

Figure 6-3 shows a more complicated example where four Xen machines (i.e. *host_20*, *host_15*, *host_17*, and *host_18*) are used to set up an emulated network with four subnets connected by four routers through static, asymmetric routing. To do so, for each subnet, one VM with two NICs can be created to act as a router node (i.e. *VR1 – VR4)*; VMs with one NIC can be created as non-router nodes (e.g. *10.0.20.10* and *10.0.15.10*). These four router nodes can be hosted on the Xen machines where the non-router nodes of its subnet are hosted, or they can be hosted together in a separated Xen machine (i.e. *host_R*) to achieve maximised routing ability by assigning special hardware resources. Note that, by "asymmetric routing", it means the path of a ping packet from 10.0.15.10 to 10.0.20.10 goes via $VR_1 \rightarrow VR_4$; while the acknowledgement from 10.0.20.10 to 10.0.15.10 goes via $VR_4 \rightarrow VR_3 \rightarrow VR_2 \rightarrow VR_1$.

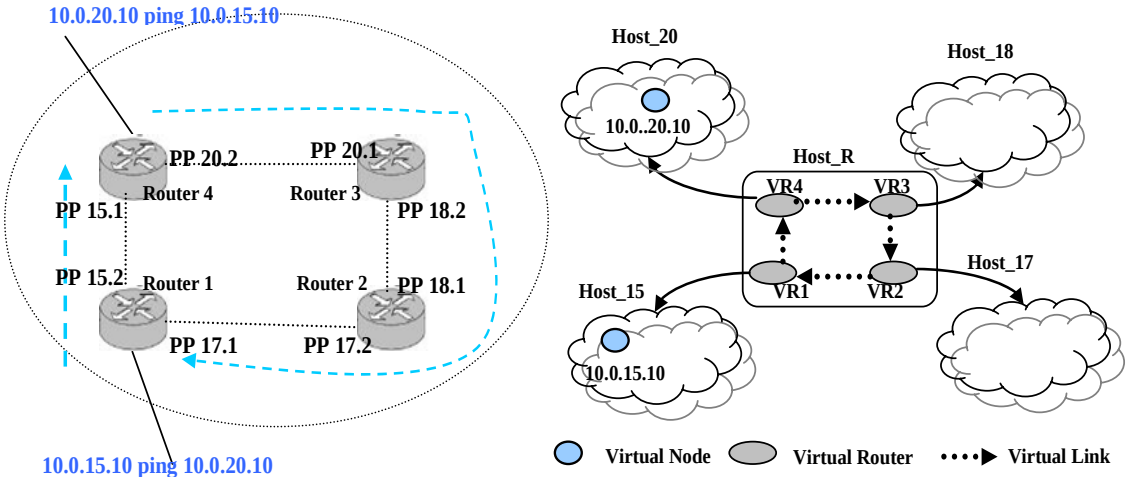


Figure 6-3: An example of emulated network with statically configured routing

6.2.2 Xen-based Emulation Framework

The possibility of using Xen to build up emulation network has been discussed in the previous section with examples. Note that in the example shown in Figure 6-3, routing is statically configured on virtual routers, and the topology is small and simple.

In contrast, to evaluate OverMon, the emulated networks are larger and more complicated, with network nodes configured as routers running the OSPF routing protocol. As a result, to thoroughly examine the performance of OverMon under different network settings, multiple experiments must be conducted and repeated with varying network size and topology, which leads to corresponding changes to router interface configuration, OSPF daemon setup, as well as application level run-time parameters. All these configuration tasks are laborious, error-prone and time consuming if done interactively through standard command line interfaces. Therefore, a configuration toolkit is required.

This toolkit is an interactive console programme written in the Python programming language and the Linux Shell script language. The system infrastructure is shown in Figure 6-4, in which the toolkit takes a topology file (generated by a topology generator) as input, and generates the configuration files and commands of the resulting emulation network, accordingly. The toolkit can be functionally divided into two parts, working at different levels: one works at the Xen host level and the other works at the VM level. The major functionalities of each level, together with the NAT configuration that plays an important role in setting up the emulated network automatically, are explained in the

following.

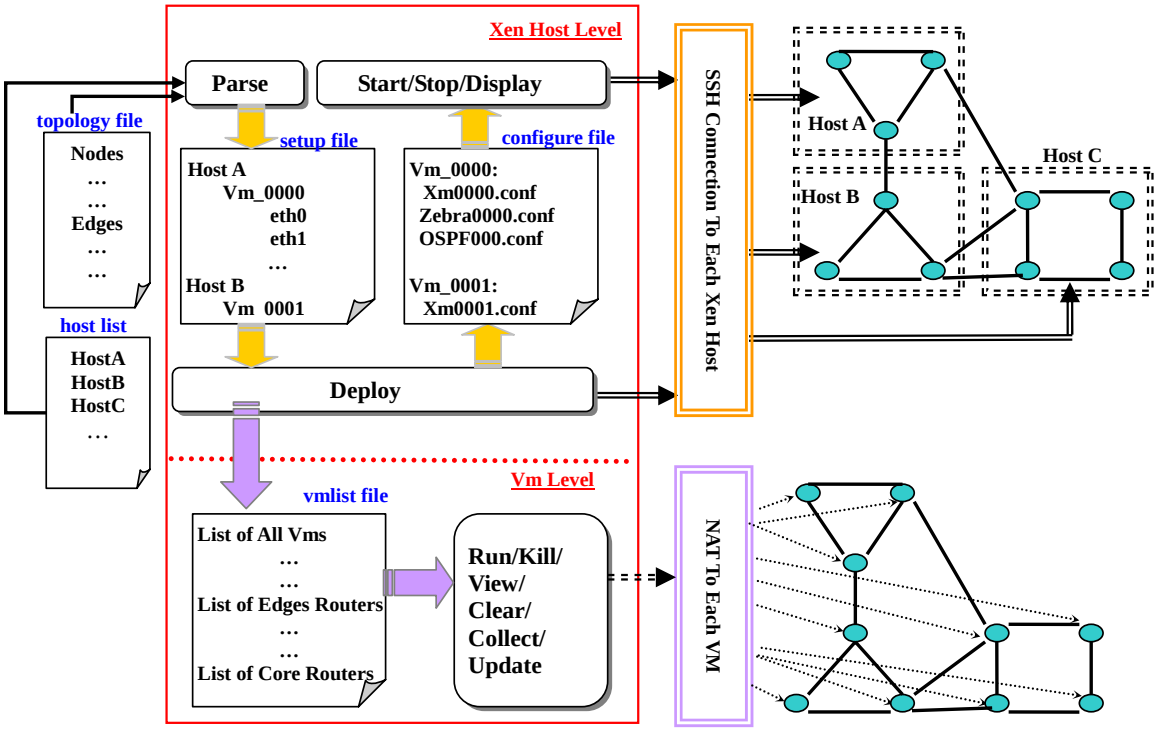


Figure 6-4: The framework of Xen-based emulation

Xen Host Level Operations At the Xen host level, the input *topology file* defines the topology of the network as a list of nodes and a list of edges connecting network nodes; additionally, the input *host list* defines the available hosts running Xen. These inputs are firstly *parsed* into a *setup file* in which network nodes of the topology file are translated into a list of VMs; the edges between nodes are mapped into IP addressed network interfaces on each VM; and these VMs are evenly assigned to available Xen hosts as specified in the host list. Next, for each VM, *configuration files* to set up the guest domain on the appropriate Xen host, as well as to set up the Zebra [ZebraWeb] and OSPF daemons on each VM, are generated and *deployed* via a SSH connection to the remote Xen host. Once the deployment of configuration files is successful, further commands can be issued, such as VM start/shutdown and display summary information of running VMs.

VM Level Opeartions At the VM level, the input is the *vmlist file* generated by the deploying process at Xen Host Level. The vmlist file specifies the list of all VMs; it also distinguishes core routers and edge routers, depending upon the number of network interfaces configured into each VM. Once the VMs are successfully started, OverMon-

relevant commands can be issued through NAT (Network Address Translation) directly to the appropriate VMs e.g. for edge router VMs to install and run OverMon software, to collect and clear evaluation log files.

NAT Setup Since the toolkit is run at a Linux machine *kettle* on the departmental network with a public IP address *130.209.241.124,* while the emulated network is constructed on subnets *192.168.0.0/16*, to access these VMs, the normal approach is via the control console of the Xen daemon at the Xen machine where the VM is hosted. This is not an efficient approach when the size of the emulated network is large. Another approach is to assign public IP addresses to emulated network nodes, but this is not practical due to a shortage of public IP addresses in the department. Therefore, to set up the emulated network nodes in a controlled and automatic manner, NAT is needed to connect the two islands of IP addresses. During the *deploy* process at Xen Host Level, the toolkit generates the NAT configuration file automatically and sets up NAT on the first VM; NAT starts to work when the first VM is successfully started. After a short stabilisation period during which the network construction is completed, further control and management commands to all other VMs can be issued.

6.3 Experimental Network Setup

Physically, the evaluation environment setup for OverMon is constructed on ten blade servers which are connected by a Gigabit switch, each with two Intel® Xeon 3.00GHz CPUs and 2GB memory. A number of virtual domains (i.e. domU) are constructed on each blade server; the VMs on all blade servers collectively form an emulation network.

Xen version 2.0 with XenoLinux version 2.6.11.10 are installed and running on both dom0 and domUs. Each domU acts as a (virtual) router and is configured with *mMB* of RAM; the memory left for dom0 is calculated as $mem_{dom0} = (2048 - m \times n)MB$, where n is the number of domUs running on the blade server. As can be seen, the memory left for dom0 is a function of the number of domUs and the memory each domU assigned. On the one hand, each Xen blade server is expected to run as many virtual routers as possible; on the other hand, dom0 serves an important role in switching network traffic, as well as keeping each domU monitored and managed. Therefore, with the current hardware configuration, this balance is maintained by assigning 368 MB to dom0 and 210MB to each domU, i.e. at most 8 domUs can be configured on each blade server.

By using the toolkit as presented in the previous section, the remainder of this section

describes the features of emulated networks that are used in evaluation. By *features*, it includes the topology model and the OSPF configuration.

6.3.1 Topology Model

Normally, to study a protocol, the performance of the protocol is strongly related to the topology that is used. For example, a delay-sensitive protocol may perform well in topologies that exhibit good delay properties e.g. non-hierarchical networks. Currently, topology models that are used for network research can be broadly classified into three categories, summarised as follows:

- Random: this type of model basically refers to a topology in which nodes are randomly distributed over a Cartesian coordinate system; the interconnection of the nodes follows a probability model. A representative example of this type is presented by Waxman [Waxman98], in which, the probability function that an edge exists between any two nodes, u and v, is given by the following probability function

$$P(u,v) = \beta \exp \frac{-d(u,v)}{L\alpha}$$

  Where $d(u,v)$ is the distance between the two nodes, L is the maximum possible distance, and α and β are parameters in the range $0 < \alpha, \beta \leq 1$. Larger values of α increases the proportion of longer edges to shorter edges, while large values of β increases the average node degree.

- Power-law: this type of model is based on the observations that node degree in the AS-level topology of the Internet is closely related to a set of power laws [Faloutsos99]. A representative example of this type is presented by Barabasi and Albert [Barabasi99], in which the probability $P(\kappa)$, that a node in the network is connected to κ other nodes is bounded, decaying as a power law

$$P(\kappa) \sim \kappa^{-\tau}$$

  where $\tau = 2.3 \pm 0.1$. There are two possible causes for the emergence of a power law in the frequency of out-degrees in a network topology: incremental growth and preferential connectivity. The former refers to growing networks that are formed by the continual addition of new nodes, resulting in a gradual increase in the size of the network; the latter refers to the tendency of a new node connecting to existing nodes that are highly connected or popular.

- Transit-Stub: this type of model models a network as a layered, hierarchical graph consisting of stub domains inter-connected by transit domains. Routers in the network are organized into logical domains, or collections of nodes. Nodes within a domain tend to be fairly interconnected within the domain, but rarely connect to nodes outside of the domain. Domains themselves are then classified into two types: transit domains and stub domains. Nodes in a stub domain are typically endpoints in a network flow — network traffic either originates at, or is destined for, a node in a stub domain. Nodes in transit domains are typically intermediate points in a network flow — traffic is typically just passing through [Zegura96].

For OverMon, the topology that is used for evaluation purpose should possess realistic features of today's network as much as possible, since such features can reflect how well OverMon performs in real-world scenarios. From this point of view, the transit-stub model has been chosen – although OverMon is to be deployed within a single AS domain, given that an AS network is very likely to be organized into multiple LANs, this model suits OverMon best. Regarding the model at each lower level, the model can be chosen from either random or power-law models.

OverMon uses the topology generator BRITE [BRITEWeb] to generate such two level network topologies. Table 6-1 lists the two topology modes used in OverMon, namely dense mode and sparse mode. The main differences between them include the average degree of nodes in the network, and the model of how nodes are placed on the plane. As shown in later sections, in each mode, by varying network size and the proportion of edge routers amongst total routers, the performance of OverMon is examined.

Table 6.1: The network models used in evaluation experiments

Mode	**Transit (Barabasi)**		**Stub (Waxmon)**				**Avg. Degree of All**	**Max Degree of Edge Router (ER)**
	m	*Node Placement*	α	β	*m*	*Node Placement*	$d_{average}$	$d_{ER\_\max}$
Sparse	2	Heavy Tailed	0.15	0.2	2	Random	≈ 4	≈ 4
Dense	3	Random	0.15	0.2	3	Heavy Tailed	≈ 6	≈ 5
Note: *m* here is an integer (>1) standing for the number of links per new node.								

6.3.2 OSPF Configuration

Once the topology generator generates the topology information, i.e. the number of nodes, the number of edges, and the connection relationship between nodes, this topological information needs to be mapped into network setup. In this setup, each link out of a router is mapped to one of the router's network interfaces with an IP address, as well as a cost (or weight) value that is used by OSPF routing daemons to calculate shortest paths and the corresponding routing tables.

In OverMon, each edge between two nodes on the topology map is taken as a subnet segment; in other words, each subnet segment connects two routers. By configuring OSPF daemon this way, to construct such dynamically built emulation networks, complexity is mitigated while generality is not lost.

For the cost value of each router's interfaces, they reflect the output side of each router interface, i.e. it is associated either with the intra-area distance between two routers, or the externally derived routing data (e.g. the BGP-learned routes).

The principle of configuring cost value is that the lower the cost is, the more likely the interface is to be used to forward data traffic. Normally, this cost is configured by the system administrator and computed by dividing the reference bandwidth (in kbps) of an interface with the configured bandwidth of the interface. For example, in practice, by default, an interface of a Cisco router is assigned a weight value proportional to the inverse of the bandwidth of the associated link.

In OverMon, without losing generality, the weight value is configured as follows: for each router, starting from eth0, the cost is 10 and increased by 1 orderly for the rest of other interfaces. Note that this is just a simple rule of weight setting. Certainly the weight for each interface can be the same, i.e., if all weights are set by a unit value, the weight of a path is the number of hops in the path. Or, weight setting can follow an advanced assignment model [Ericsson02], with the objective of minimizing network congestion. Given that the evaluation process in OverMon is to conduct a *relative* comparison of variant overlay level algorithms and protocols, and they are deployed upon the same emulated network topology formed by the OSPF daemon, the weight setting model in OverMon will not introduce any bias in evaluation.

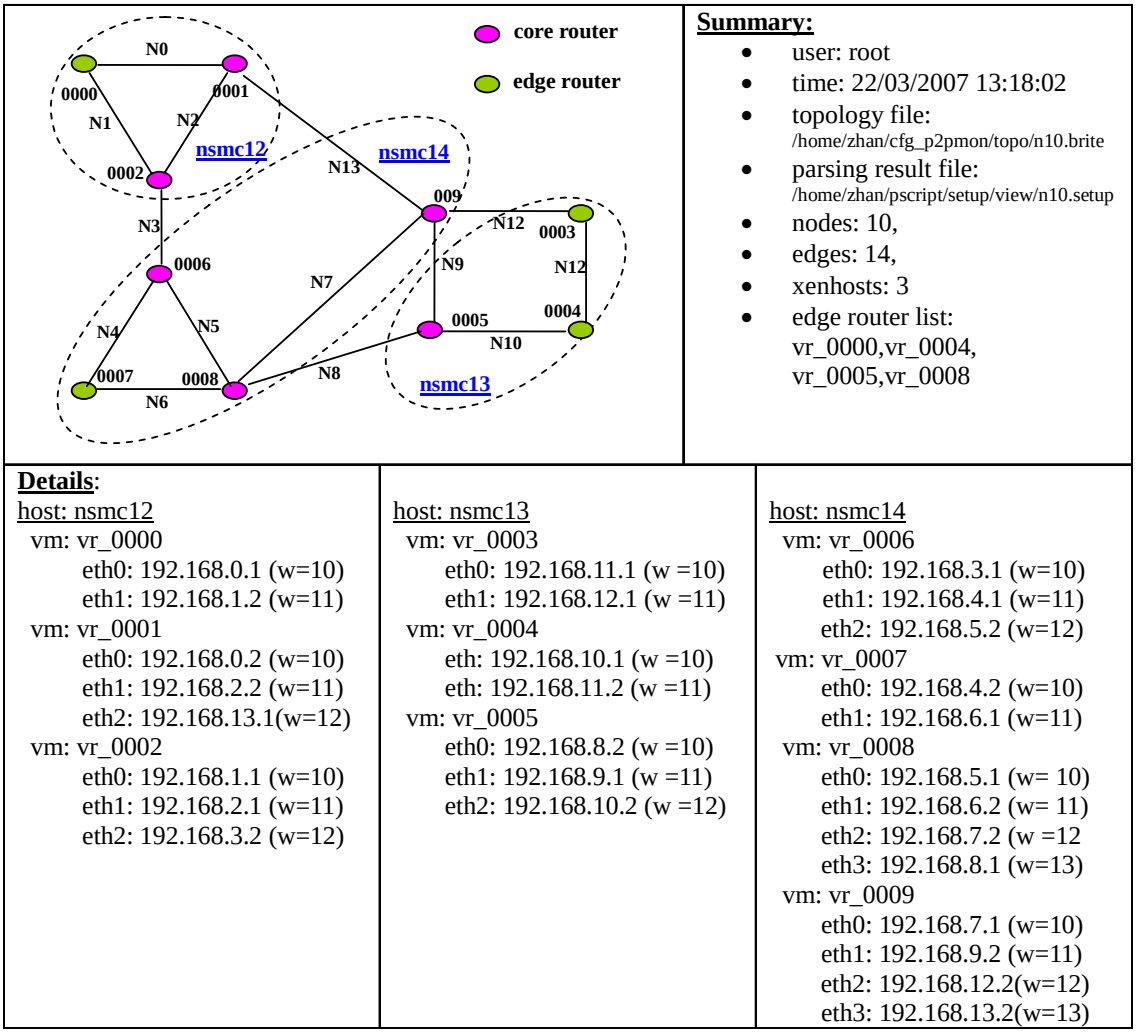


Figure 6-5: An example of transforming a topology map into OSPF network configurations

Figure 6-5 illustrates an example of how a topology map consisting of 10 nodes is transformed into an OSPF network configuration file. After the transformation, there are 14 network edges implying 14 network subnet segments (numbered 0, 1, etc.). Those nodes that have more than two edges are considered as core routers, while the rest are considered as edge routers.

Note that the division of core router and edge router in this example is just for illustration purposes. However, it reflects the principle of separating edge routers from core routers: those having more edges are taken as having more network connectivity, therefore are core routers; in contrast edge routers having fewer edges have less connectivity, implying they are connected by core routers. For the two network models introduced in Section 6.3.1, by setting the threshold of maximal degree for edge routers, nodes within a given topology can be easily divided into two groups: core router and edge router.

6.4 Evaluation of Control Overlay

This section presents a detailed evaluation of OverMon's control overlay. Due to the fact that OverMon's control overlay for building application level multicast trees has special features that other ALMs don't have, the strategy is to conduct evaluation thoroughly by both simulation and emulation, focusing on its efficiency in the use of the underlying network.

6.4.1 Metrics

Generally speaking, in overlay networks, network edges are direct UDP/TCP connections between pairs of nodes, and overlay packets are physically forwarded by routers, hop by hop, along the unicast path. Since overlay networks can't control how packets are forwarded in the underlying physical network, packets might be transmitted on some of the links more than once, hence extra delay might be caused compared to native unicast or IP multicast.

Since different ALM algorithms have various properties that make them suitable for different applications, it is difficult to compare the value of these metrics without taking into account factors such as underlying network performance and application level requirements. Furthermore, there often exists a trade-off between performance and overhead: multicast members periodically exchanging updated state information, which

causes control traffic but at the same time affords flexibility and resilience in optimal path selection and failure recovery.

To evaluate these overheads, the following metrics are commonly used for measurement of ALM performance:

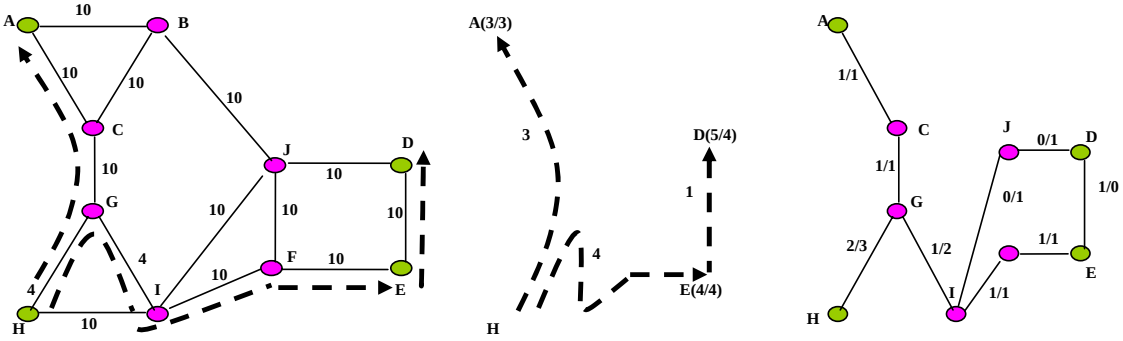
- **Transmission Cost**: defined as the average cost of sending a packet from one group member to the rest of the group. It can be measured by summing up all weight values of any parent to its children along the unicast paths. In other words, this includes the cost of multiple traversals on some of the network links. Note that if it refers to the count of packets and corresponding bytes seen at application level, the total cost of multicast and of unicast is equal; however for the root node, the packets sent out in multicast are greatly reduced; hence the bottleneck issue at the root node is mitigated.
- **Link Stress:** defined as the total number of identical copies of a packet travelling over a single physical link, i.e. the duplicate packets on the links. For network level IP multicast, the link stress is always one; for overlay level multicast, the multicast packets are forwarded along the unicast paths, therefore a router may receive and send data over the same network interface, causing duplicate packets to be transmitted.
- **Link Stretch**: defined as the ratio of the length along the overlay path relative to the network level unicast path between two nodes. This pair-wise metric can be measured either by hop or by delay, and essentially captures the additional distance that a packet must cover relative to the unicast path. The shortest path tree has a link stretch of one, and typically application level overlay multicast has a stretch greater than one.

Note when link stretch is measured by delay, it is also referred to as RDP (Relative Delay Penalty); and the measurement result closely relates to the run-time network traffic level. In that case, if the clock on each node is well synchronised, no doubt this process assists in determination of real world delay of overlay packets. However, even though Xen is designed to have each virtual domain (i.e. domU) synchronized with domain 0, and each domain 0 can be synchronized with a higher level time source, bias might be introduced depending upon whether the two virtual domains are physically hosted by one or by two machines. Given that OverMon is not a delay-sensitive application, (e.g. real-time video/audio broadcasting), the delay metrics are approximately measured by hops, i.e. by link stretch. Nevertheless, it is interesting to unveil the relationship between measurement

delay and measured hops for a specific emulation environment such as the one built upon Xen.

In Figure 6-6, the ten-node topology shown earlier in Figure 6-5 is used to illustrate the evaluation metrics. For demonstration purpose, it is transformed into an undirected graph with each direction of an edge having the same weight value of w, so that the weight value marked on each undirected edge in the graph stands for $2w$. In Figure 6-6 (a), an ALM tree is formed from edge routers (i.e. labelled by A, D, E, H), and rooted at H. The abstract graph representation of the overlay network is given in Figure 6-6 (b). The links traversed by ALM and by unicast are shown in Figure 6-6 (c). As can be seen,

- The total transmission cost for ALM is (12+14+5) =31, while (12+14+17)=43 for unicast (with the path of *H-G-I-F-E* being saved in sending packet to D in ALM);
- The three receivers (i.e. A, D, E) are reached by (3,4,5) hops respectively in ALM, and (3,4,4) hops respectively in unicast; thus average ALM stretch = (3/3 + 4/4 + 4/5)/3 = 1.08;
- The link (*H*,*G*) is used twice in ALM, three times in unicast; and the link (*G*,*I*) is used once in ALM, and twice in unicast. Totally, 7 links are used in ALM, while 9 links are used in unicast, thus average ALM stress = 8/7=1.1428572, average Unicast stress = 11/9=1.22.



(a) Multicast session comprising four edge routers

(b) Resulting Stretch. Nodes are marked by ALM/ Unicast

(c) Resulting Stress. Edges are marked by ALM/Unicast stress

Figure 6-6: An ALM overlay network built on a ten-node topology with multicast session comprising three receivers

As discussed in the previous chapters, OverMon's use of the network differs from other overlay applications in many ways. The most obvious difference is that in OverMon,

members do not dynamically join/leave a session, and there is no time for the tree to be optimised gradually during the session. Therefore it is difficult to find a similar ALM system to do a head-to-head comparison. Probably, the most relevant work is the evaluation methodology adopted by TAG (Topology Aware Grouping) as proposed in [Kwon02], which follows the Chuang-Sirbu Law [Chuang98]. In the Chuang-Sirbu Law, a cost-based formula is used for (IP level) multicast pricing to state a normalized multicast tree cost:

$$\frac{Lm}{\overline{Lu}} = N^k \qquad (1)$$

Where Lm : Total length of multicast distribution tree;

$\overline{L_u}$: Average length of unicast routing path;

N : Multicast group size;

k : Economies of a scale (EoS) factor, ranging between 0 and 1.

This means that the cost of the normalized multicast tree is a dimensionless parameter-although $\overline{L_u}$ is network-specific and influenced by topological factors such as the number of nodes and links in the network, average node degree, network diameter, etc, the cost value, however, should be relatively static and the cost of a multicast tree varies as the 0.8 power of the multicast group size, i.e. $k = 0.8$. This work is further validated by [Chalmers01], in which $k = 0.6 \sim 0.7$ are proposed for smaller session sizes between 20 and 40 receivers, $k = 0.8$ for 150 receivers and $k = 0.9$ for large groups with 1,000 receivers or more, with a slight difference from [Chuang98] in considering the last-hop end hosts. In [Kwon02], overlay multicast networks are characterized by using the number of hops as the metric, and showed that $k = 0.9$ applies to small session size, which is comparable to IP multicast cost proportional to $k = 0.6 \sim 0.8$ in [Chuang98, Chalmers01].

As a result, taking $k = 0.6 \sim 0.9$ as the upper and lower bounds, OverMon can be evaluated by comparing against *general* IP level multicast and overlay multicast, with the expectation that the curves of OverMon closely approximate the curves of IP level multicast, while lower than the curves of overlay multicast. This evaluation is sensible since the ultimate target for application level overlay multicast is to achieve a performance as close as possible to that of IP level multicast.

Additionally, since OverMon is taking the advantage of topology information to construct the multicast tree, it is meaningful to compare the multicast tree generated for a given

topology by OSPF snooping against the tree generated by other approaches where topology information is not available, for example, by firstly probing other nodes to obtain the distance relevant information such as delay, then by exchanging the end-to-end measurements to obtain the entire (approximate) topology information.

6.4.2 Methodology

As discussed, the evaluation of OverMon's control overlay consists of two parts: emulation and simulation. This section firstly covers the design of the emulation experiments, which are based on the emulation environment discussed in Section 6.2; then it describes the customised simulation software, which has been developed to cater for ALM in OverMon.

6.4.2.1 Emulation

The emulation experiments that have been performed are illustrated in Figure 6-7, and are explained as follows:

- With a configured network size, yielding a fixed edge router set, the multicast session consists of all edge routers.
- To make the results fair, i.e. to alleviate the issue that multicast performance is dependent upon the tree's shape, each edge router is chosen as the root node from which to initiate the multicast session.
- Firstly, the multicast packet is relayed along the multicast Steiner Tree (ST).
- Secondly, the unicast packet is relayed along the Shortest Path Tree (SPT) consisting of shortest paths from root node to each receiving node.
- Once a cycle of each node acting as the root node to initiate the multicast/unicast trees is finished, the performance is measured by averaging each session's result.
- Each experiment is repeated twice, with topology information being obtained through OSPF snooping and through pair-wise probing, respectively.

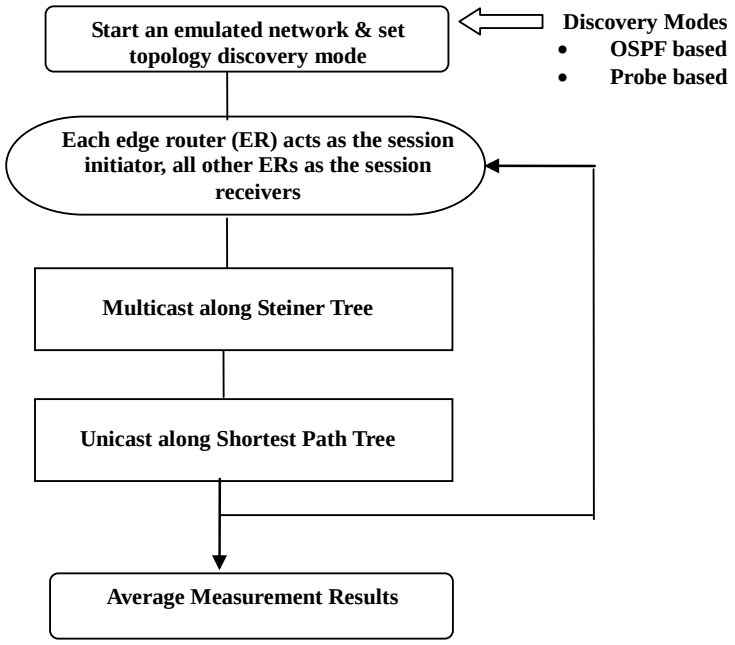


Figure 6-7: Experimental loop of emulation for control overlay

In the case of pair-wise probing, the estimate of the network topology is periodically constructed as follows:

1. Each edge router sends an UDP packet as a probe to each of the other edge routers and calculates the packet delay when it receives the corresponding ACK packet.
2. Each edge router sends this updated packet delay information, i.e. from itself to all other edge routers, to a centralized aggregator. The aggregator aggregates the reported delay measurement into a $V*V$ adjacency matrix with each row representing a source, each column representing a destination, and each cell containing the delay information from a source to a destination.
3. The aggregator sends the summarized adjacency matrix back to each edge router, where the topology graph of the network is constructed by taking the adjacency matrix as a full connection graph. The graph is directed with each cell in the matrix representing the weight of corresponding directed edge between two vertices, i.e. the distance (by delay) from one edge router to another edge router.

Note that in this pair-wise probing approach, since core routers are not involved in the probing process, the resulting topology graph is just a partial view of the whole network topology; it reflects the end-to-end distance between a pair of edge routers, i.e. the result

of packets being routed along the shortest path between two edge routers in the physical network.

The configuration of periodically sending the topology relevant packets as discussed above is as follows:

- The UDP probe packets are sent at a *probe interval* I_{probe}, which by default is 10 seconds, identical to the *hello interval* of OSPF Hello packet in the same network environment[16].
- The packets of delay information from an edge router to the centralized aggregator are sent at an *update interval*, which by default is four times the value of the probe interval herein 40 seconds, i.e. $I_{update} = \omega_1 * I_{probe}$, and $\omega_1 = 4$. The setting of update interval is similar to the *dead interval* in OSPF, which by default is four times the value of the *hello interval*.
- The packets of summarized adjacency matrix from the centralized aggregator to edge routers are sent at a *summary interval*, which by default is two times the value of the update interval herein 80 seconds, i.e. $I_{summary} = \omega_2 * I_{update}$, and $\omega_2 = 2$.

Therefore, by default, the extra overheads caused by probing include three parts:

- $B_{ping} = P_{ping} * N_{edge} * (N_{edge} - 1)$ bytes every 10 seconds, as the cost of probe packets, where N_{edge} denotes the total number of edge routers and P_{ping} is the size of UDP pinging message in bytes;
- $B_{update} = P_{update} * N_{edge}$ bytes every 40 seconds, as the cost of update packets sent by edge routers to the centralized aggregator;
- $B_{summary} = P_{summary} * N_{edge}$ bytes every 80 seconds, as the cost of summary packets sent by the centralized aggregator to edge routers.

Obviously, this can be the worst case to calculate probing overheads, since rather than probing *each* of the other edge routers in the network, a subset of the neighbouring edge routers can be probed; also larger intervals can be used to send probes, updates and summaries. In fact, all these solutions trade overhead for quality, since the probed information is based on imprecise information which consequently could cause

[16] In the OSPF protocol, the Hello packets are sent at a configurable interval (in seconds). The defaults are 10 seconds for an Ethernet link and 30 seconds for a non broadcast link.

suboptimal routing and eventually reduce the total network utilization.

To deploy the emulated network topology, the Xen-based emulation toolkit as discussed in Section 6.2 is run on a Linux machine named *kettle*. Instead of issuing VM level commands each time directly from *kettle* to the virtual nodes via NAT, a set of scripts have been developed to run experiments in a *batch* style.

In this batch style running, the whole experimental loop illustrated in Figure 6-7 needs OverMon software to be initiated only once. Run-time configurations, such as the communication mode of multicast or unicast, the role of each node in acting as initiator or receiver, are configured dynamically via RMI calls. Furthermore, for each *experimental iteration*, in which one of edge routers is chosen as the root node and all the others act as receiving nodes, the logged information is stored in separate trace files for each virtual node, and is gathered and stored locally on *kettle* in a hierarchical structure.

The benefit of using this batch style running is that, given an emulated network setup and a chosen mode (OSPF snooping mode or UDP probing mode) for discovering topology information, experiments can be continuously iterated, thus the system does not need to be restarted from scratch each time, and as a result, does not have to wait for the OSPF configuration to stabilise. In addition, the separated archives of raw trace files can ensure the repeatability and verifiability of the evaluation calculations.

6.4.2.2 Simulation

Since the emulation experiment discussed above is time consuming (due to averaging of multiple experiment iterations), and inherent lack of scalability, customised simulation is a complementary means of evaluation. By “customised simulation”, it refers to running simulating software that is specifically developed for ALM algorithms in OverMon. Specifically, it refers to the fact that it does not take timing into account and ignores dynamic changes in network conditions; herein it is distinct from a discrete event based simulation in that it statically parses the topology files and executes the tree construction algorithm.

In this simulation software, the topology map is formed by parsing the original static topology file generated by the BRITE topology generator, i.e. from a topology file that only specifies the nodes and the edges of the topology. Then, rather than generating Zebra and OSPF configuration files for each individual virtual router and waiting for them to form the topology incrementally via OSPF packet exchanging, the simulation software directly transforms the nodes of the topology graph into OSPF routers with

corresponding router ID and IP-addressed interfaces, and edges of the topology graph into weighted network links connecting the OSPF routers.

Since the resulting topology map is in the identical data structure as used by the emulation, the implemented code can be reused in taking this topology map as the input to compute the multicast/unicast tree. Then, instead of physically sending overlay construction packets out from sockets, the simulation software writes the generated multicast/unicast tree locally into a log file; statistics can then be generated using logged information regarding features of the multicast/unicast tree.

Note that as discussed in Section 6.4.1, for delay measurement in OverMon, the metric of link stretch, (rather than time stamped RDP), is used. Therefore by using this simulation software, the emulation loop as shown in Figure 6-7 can be fully repeated and the identical evaluation can be performed. Furthermore, larger size of networks with a broader range of network configurations and run-time parameters can be quickly set up for testing purpose, such that the evaluation process can be accelerated.

6.4.3 Results

To present the evaluation results, this section is divided into three parts, namely emulation result, verification of simulation, and simulation result. For each part, the objectives are different, and are summarized as follows:

1. Emulation Result
   - For a given approach to obtain network topology information, compare the performance of overlay multicast against unicast;
   - Compare the performance improvement of overlay multicast when topology information is obtained through OSPF snooping and through UDP-based probing.
2. Verification of Simulation
   - Verify that the simulation software is correctly implemented.
   - Verify that the topology information snooped by OSPF snooping is accurate.
3. Simulation Result
   - Investigate the variance/invariance of OverMon performance with varied topology models and runtime parameters.

6.4.3.1 Emulation Results

The networks used in the emulation experiments follow the sparse model as specified in Table 6-1; the run-time configurations are summarised below in Table 6-2. Note that the values in the column of “ER_Max” stand for the maximal degree for edge routers. They are set up to distinguish edge routers from core routers, as explained in Section 6.3.2. Correspondingly, the values in the column of “# of Edge Router” stand for the number of edge routers, which are calculated as the result of setting up “ER_Max”..To achieve a consistent proportion of edge routers for variant network sizes, small adjustments of taking some edge router as core routers are made, thus the proportion of edge routers among total routers is 65%.

In each emulation experiment, when a virtual node participates in a multicast session, it logs information about each overlay message it sends out. The information includes the message’s type (e.g. multicast or multicast ACK) and corresponding size in bytes. For a root node, when the multicast session that it initiates successfully finishes, i.e. when the root node receives ACKs from all its children in the multicast tree, it also logs the multicast tree topology that it has computed and relayed. All of the logged information is gathered by *kettle*, and verification is performed by comparing the assembled data from each individual node against the logged data from the root node. After the logged data are validated, statistics regarding transmission cost, link stress and link stretch, are calculated.

Table 6.2: Run-time configurations for emulated networks

# of Nodes	# of T-S	# of Edges	# Degree				# of Edge Router
			Min	Ma	Avg	ER_Max	
15	3-5	25	3	4	3.33	3	10
25	5-5	43	3	4	3.44	3	15
32	4-8	67	3	6	4.19	4	21
40	5-8	84	2	6	4.25	4	27
54	6-9	116	2	7	4.30	4	35
70	7-10	151	2	8	4.31	4	47
77	7-11	165	3	8	4.29	4	52

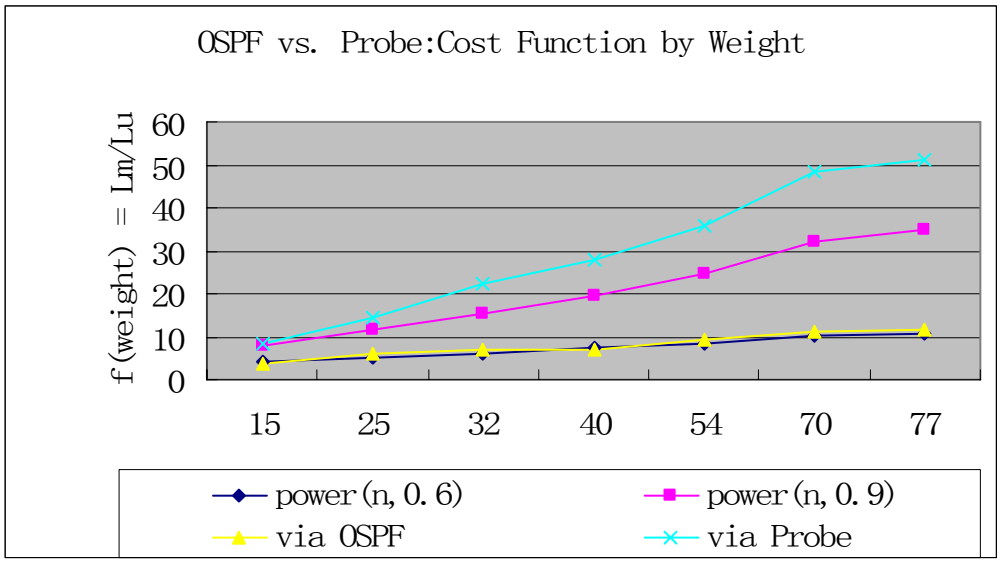
The results of measuring the metrics as discussed in Section 6.4.1 are shown below.

Transmission Cost

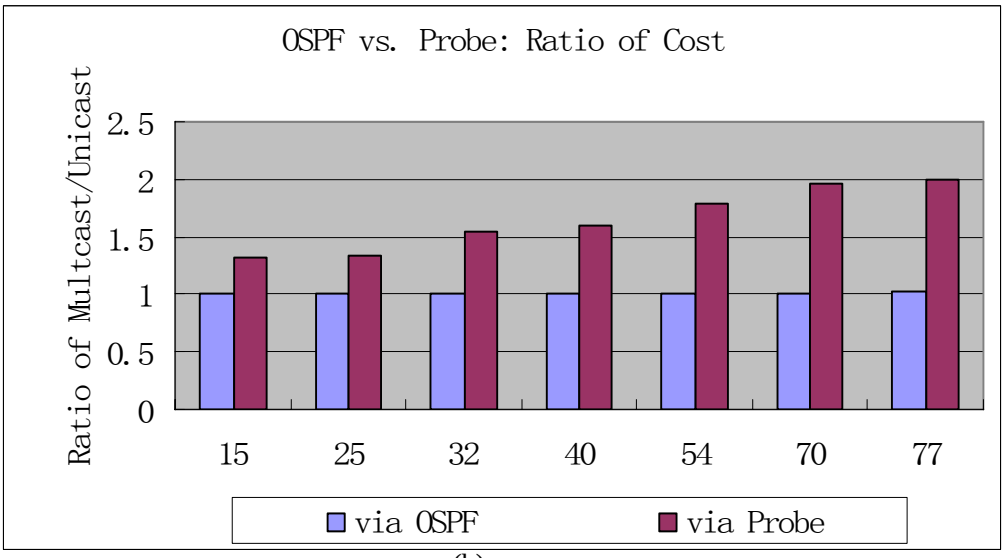
In Figure 6-8 (a), the transmission cost of multicast tree based on OSPF snooping and on UDP-based probing is normalized by $\frac{Lm}{\overline{Lu}} = N_{edge}{}^{k}$. The y-axis denotes the value of $\frac{Lm}{\overline{Lu}}$, and the x-axis shows the total network size including both core routers and edge routers - among them, about 60 - 71%[17] are chosen as edge routers participating in the multicast session. Note that N_{edge} is the number of edge routers that is counted to calculate N^k in equation (1); and k is set to 0.6 and 0.9 respectively, representing the performance of *general* IP multicast and overlay multicast. The value of Lm is computed by simply summing the edge costs (i.e. weight values) of all links that make up the multicast tree; and $\overline{L_u}$ is computed by summing the edge costs of all links that make up the unicast paths and dividing it by the number of receiving nodes (i.e. $N_{edge} - 1$).

In particular, in the case of UDP-based probing, since the multicast tree is computed from the pair-wise end-to-end delay measurement between edge routers, the total network level hops is computed by *mapping* each application level multicast/unicast hop to the corresponding shortest path along the physical network topology. Without otherwise stated, this mapping applies to the calculation of stress and stretch in the following subsections.

[17] Since it is not easy to acquire accurate information regarding the percentage of core routers and edge routers in a commercial ISP network, admittedly these figures are intuitively set up.



(a)



(b)

Figure 6-8: Control overlay evaluation –transmission cost in emulation

Figure 6-8 (a) demonstrates that OSPF snooping based multicast has lower transmission cost than general overlay multicast does, as its curve closely approximates the curve of IP multicast (i.e. power(n,0.6)); while in the case of multicast trees built upon UDP-based probes, the curve is always higher than normal overlay multicast (i.e. power(n,0.9)).

In Figure 6-8 (b), the transmission cost is normalized by calculating the ratio between total costs of overlay level multicast and overlay level unicast. As seen from the figure, for each network size, the OSPF based approach consistently achieves smaller ratio of transmission cost than the probe based approach does: the curve of the OSPF snooping

approach is steadily around 1.0; while in the curve of the probe approach, the largest ratio is approaching to 2.0 when network is 70.

As these results imply, the OSPF snooping approach achieves better performance than the probe approach does; it causes less bandwidth usage and the level of its performance is close to the level that IP multicast can achieve.

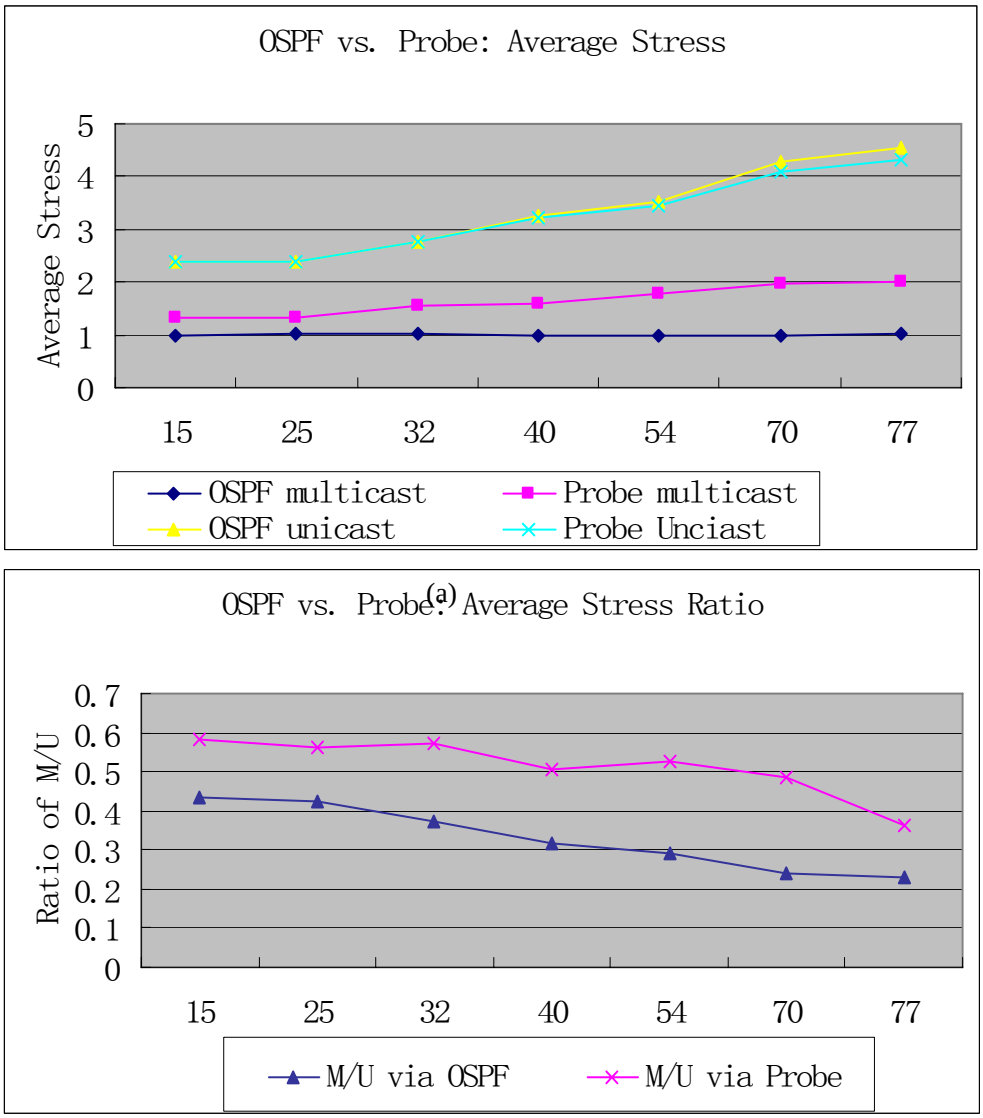
Link Stress

The link stress metric is plotted in Figure 6-9, with the x-axis denoting varying network size and the y-axis denoting the average link stress in plot (a) and the ratio of average link stress between multicast and unicast in plot (b).

The reason of taking average stress, rather than the max stress, is that obviously for unicast, the stress at the root node is much higher than multicast and is direct proportion to the total number of receiving nodes. Therefore, to make the comparison unbiased, the average stress is used; it is calculated by summing up total stress on each link of the multicast tree (or unicast tree), then dividing this value by the total number of links that the multicast tree (or unicast tree) covers on physical networks.

The absolute comparisons are shown in plot (a). In the case of unicast, for both of the OSPF snooping approach and the UDP-based probing approach, the curves of average link stress increase linearly with network size, and the values are as high as 4.5 when the network size reaches to 77. In the case of multicast, OSPF snooping approach achieves average link stress consistently around 1.0; this is approximate to IP multicast by which the link stress is always 1.0, and outperforms other non-topology-aware ALM systems by which the average stress is 1.19 -1.51 in [Lao07] and 2.0 in [Banerjee02][18]. For the multicast trees constructed upon UDP-based probing, the average stress is always larger than 1.0, which means most of the network level links covered by the multicast sessions carry duplicate packets.

[18] According to the cited papers, these figures are obtained through simulation, and their functionality includes membership maintenance.



(a)

(b)

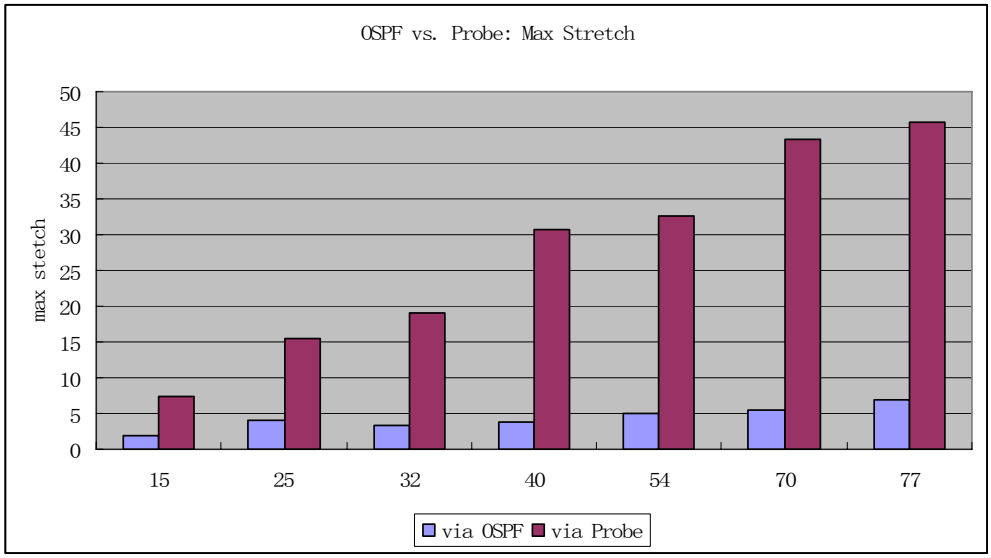
Figure 6-9: Control overlay evaluation –stress in emulation

Relative comparisons are shown in plot (b), by the ratio of average stress between multicast and unicast. As seen, for the ratio of OSPF snooping approach, the value is always less than 0.5, and always less than that of UDP-based probing; this means the OSPF snooping approach improves performance in a larger degree than the probe approach does, in reducing the duplicate packets transmitted by the underlying network.

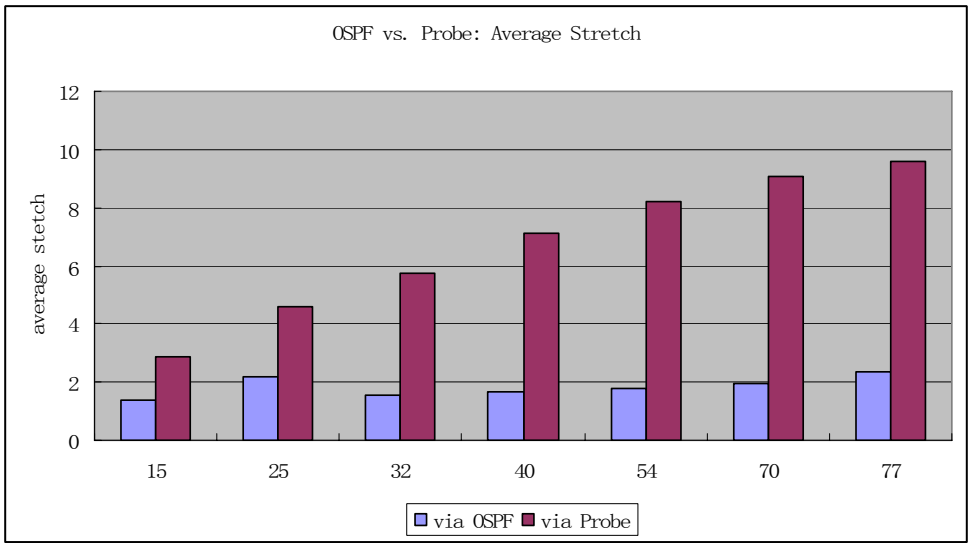
Link Stretch

The metric of link stretch is plotted in Figure 6-10, with the x-axis denoting varying

network size, and the y-axis denoting the comparison of the link stretch for multicast trees built upon OSPF snooping and upon UDP-based probing respectively. In Figure 6-10, chart (a) plots the maximum stretch, showing the worst case stretch, and chart (b) plots the average stretch, showing the majority of stretch between any two nodes on the tree when the multicast tree is rooted at different edge routers. Since the minimum stretch hovers near 1.0 for both approaches, regardless of network size, the minimum stretch is not plotted.



(a)



(b)

Figure 6-10: Control overlay evaluation – stretch in emulation

As can be seen, the probe approach produces distinctly larger stretch for both average and maximum stretch than the OSPF snooping approach does. For example, in the case of maximum stretch, the UDP probe approach can reach a stretch larger than 45 when network size reaches to 77; while at the same network size, in the OSPF snooping approach, the value a slightly larger than 7. Similarly, in the case of average stretch, the difference is obvious too: when the network size is 77, the average stretch in the probe approach is about 9.6, while 2.4 in the OSPF snooping approach.

Another character of Figure 6-10 is that the OSPF snooping approach achieves a relative stable stretch; while in the UDP probe approach the stretch linearly grows with the network size. For example, the average stretch for the OSPF snooping approach fluctuates in a small range of 1.37 ~ 2.38; while for the UDP-probe approach, this average stretch is much higher and fluctuates in a range of 2.85-9.59.

Therefore, as a summary of emulating results, 1) multicast achieves better overall performance than unicast; 2) the OSPF snooping approach performs much better than the UDP probe approach in constructing multicast trees. 3) the performance improvement is manifest even when the network is small; and 4) this improvement is achieved without introducing any maintenance traffic for probing.

The main reason behind the probe approach achieving worse performance is that the pair-wise probed information is not as precise as the topology information snooped from OSPF packets, since the probed delays only reflect partial topology information between edge routers, and are affected by network load at that moment the probes are sent, hence are likely to be out of date when they are used for the multicast tree calculation.

Another factor contributing to the lack of imprecision in probe based approach is the use of Xen to construct an emulation environment for delay-sensitive applications (i.e. by using measured delay as the input in estimating the network topology), since when two virtual nodes are hosted by one physical machine, the delay between them is very likely smaller than the case when they are hosted by two physical machines. As a consequence, the probed delay and the resulting multicast tree might be biased by the assignment of virtual nodes to physical machines, an artefact of the emulation environment. The quantification of this factor can be an interesting topic for the further work.

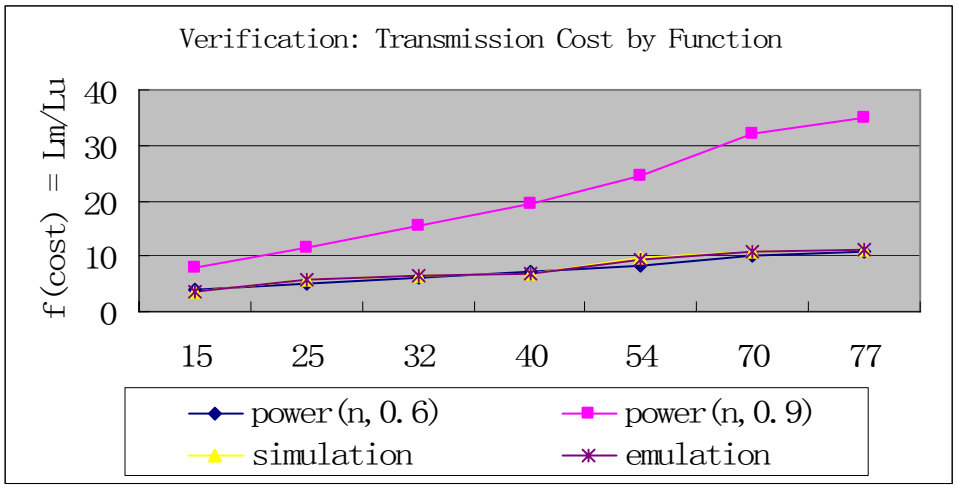
6.4.3.2 Verification of Simulation

Since the resulting topology map can be easily checked against the topology generated by the topology generator, if the simulation is performed with the same network

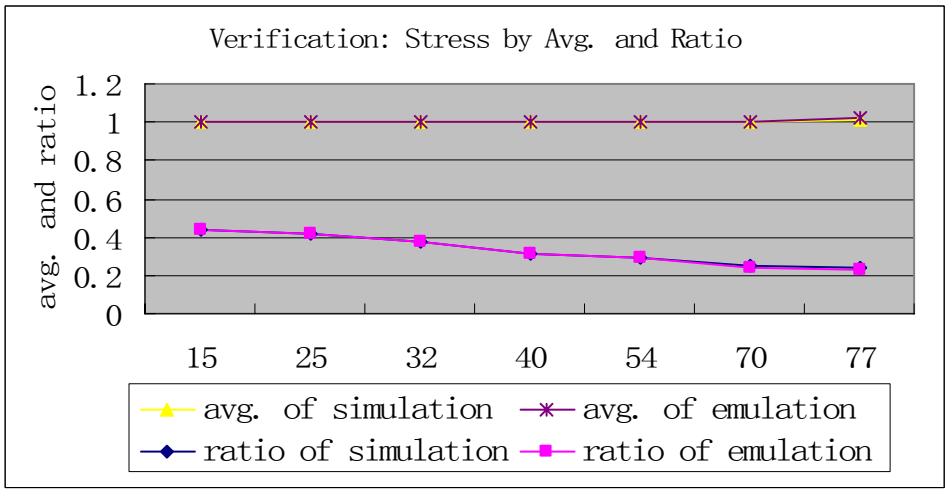
configuration and runtime parameters as in emulation experiment, and, if the results from simulation and emulation are same, the two sets of results are mutually verified: the simulation software is correctly implemented, and the network topology formed by snooping from OSPF packets is correct.

Figure 6-11 plots the head to head comparison of results from emulation and simulation. As shown in the figures, the curves are very similar, although slight differences exist between the simulation and emulation results. This might be caused by incremental topology database construction by OverMon - it constantly checks the validity and freshness of the database records, and updates the database records by deleting a stale instance before inserting the corresponding new instance; due to multi-threaded programming for tree construction and database maintenance, there exists a chance that the multicast session is initiated at the moment database records are being renewed.

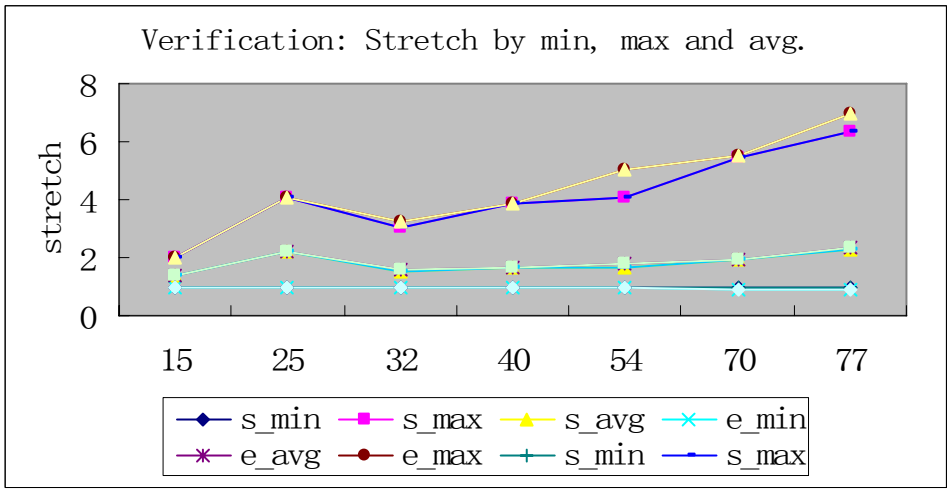
However, by using software techniques such as mutual exclusion, it is possible to eliminate the slight differences. More importantly, the verification results show that the simulation software is correctly implemented, and the topology information snooped by OSPF snooping is accurate; thus it enhances one’s confidence that simulation results for larger network sizes or different network models can be believed.



(a)



(b)



(c)

Figure 6-11: Control overlay evaluation - verification of emulation and simulation

6.4.3.3 Simulation Result

To reflect OverMon's performance on different network topologies, multiple simulations are performed on different network models, such as the sparse and the dense models listed in Table 6-1. Different run-time network configurations, e.g. network size and node degree, as well as corresponding proportion of edge routers among all nodes, are listed in Table 6-3. Note the proportion of edge routers among total routers firstly is 65%, then around 40%.

Here, the *dense* and the *sparse* refer to the relative density of the topology graph [Preiss98], which can be formalized by the following formula:

$$D = \frac{2|E|}{|V| * (|V| - 1)}$$

where |E| denotes the number of edges, and |V| denotes the number of nodes. For the sparse model as listed in Table 6-3, the average D for variant network sizes is $Avg\_D_{sparse} = 0.043$, while for the dense model it is $Avg\_D_{dense} = 0.062$; the density ratio between these two models can be calculated as below, meaning the dense mode is nearly half times denser than the sparse mode (i.e.45%).

$$\frac{Avg\_D_{dense}}{Avg\_D_{sparse}} = \frac{0.062}{0.043} \approx 1.45 .$$

Table 6.3: Run-time configurations for simulated networks

Mode	# of Nodes	# of T-S	# of Edges	# Degree			# of Edge Router	
				Min	Max	Avg	65% $d_{ER\_\max}$ = 4	40% $d_{ER\_\max}$ = 4
Sparse	54	6-9	116	2	7	4.30	35	22
	70	7-10	151	2	8	4.31	47	28
	77	7-11	165	3	8	4.29	52	30
	90	9-10	195	2	8	4.3	61	36
	99	9-11	213	2	9	4.3	70	40
	120	10-12	257	2	8	4.28	80	48
	140	10-14	297	2	10	4.24	96	56
	160	10-16	337	2	10	4.21	113	64
	180	10-18	377	2	10	4.19	120	72
	200	10-20	417	2	12	4.17	133	80
Mode	**# of Nodes**	**# of T-S**	**# of Edges**	**# Degree**			**# of Edge Router**	
				Min	**Max**	**Avg**	**65% $d_{ER\_\max}$ = 6**	**40 % $d_{ER\_\max}$ = 5**
Dense	54	6-9	159	4	8	5.89	32	19
	70	7-10	216	3	9	6.17	40	23
	77	7-11	244	3	10	6.33	41	24
	90	9-10	280	3	10	6.22	53	27
	99	9-11	310	3	11	6.26	56	32
	120	10-12	382	3	10	6.36	70	41
	140	10-14	444	3	11	6.34	78	46
	160	10-16	504	3	13	6.3	93	69
	180	10-18	564	3	14	6.27	107	74
	200	10-20	624	3	13	6.24	124	8

Transmission Cost

The evaluation of transmission cost normalized by the Chuang-Sirbu Law is plotted in Figure 6-12, with plot a–d representing the result of simulating sparse and dense network models with edge router approximately proportional at 65% and 40% respectively.

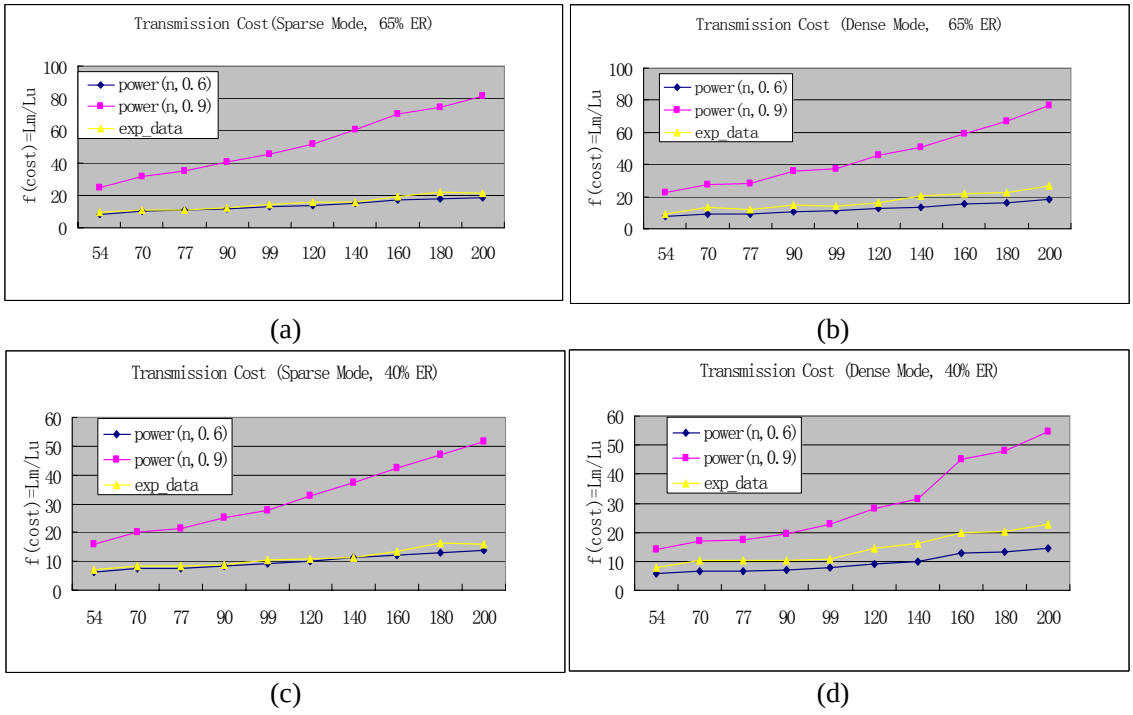


Figure 6-12: Control overlay evaluation - transmission cost in simulation by function

As shown in the figures, similar characteristics of transmission cost are exhibited for the four cases and the performance levels are all close to IP level multicast. For different network models but same proportion of edge routers, such as plot (b) to plot(a), and plot (d) to plot (c), the gap between the OverMon curve and that for the curve of power(n,0.6) is slightly different: for the dense model, where the density is about 45% higher than the sparse mode, the improvement of transmission cost is less. If the gap is calculated by the formula of

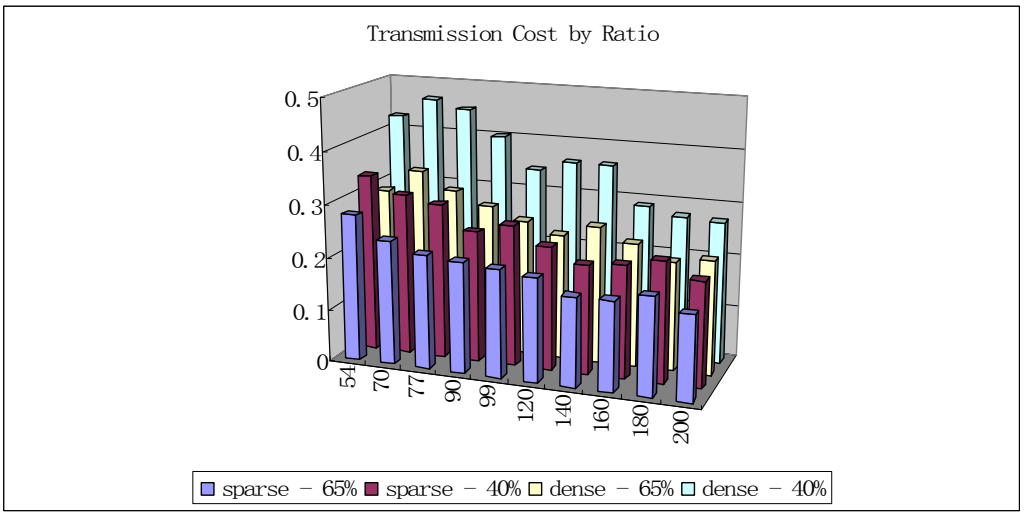
$$TC_{gap} = \frac{\sum \frac{R_n - power(n,0.6)}{power(n,0.6)}}{k}$$

Where n: The size of each multicast session

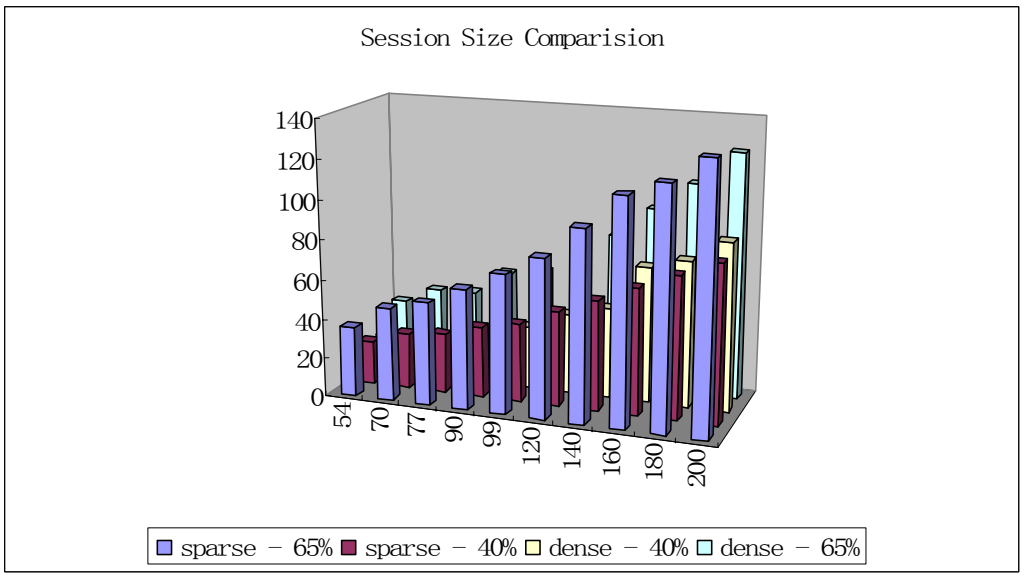
R_n : The simulation result for each n

κ : The times of running experiments (e.g. κ = 10 in this case).

In the case when the proportion of edge routers is 65%, TC_{gap} = 11.3% for the sparse model and TC_{gap} = 35.7% for dense model; similarly, when the proportion of edge routers is 40%, TC_{gap} = 11.6% for the sparse model and TC_{gap} = 50.5% for dense model.



(a)



(b)

Figure 6-13: Control overlay evaluation - transmission cost in simulation by ratio

In Figure 6-13, the ratio of total transmission cost between multicast and unicast is shown in figure (a), and corresponding multicast session size is shown in figure (b). It shows

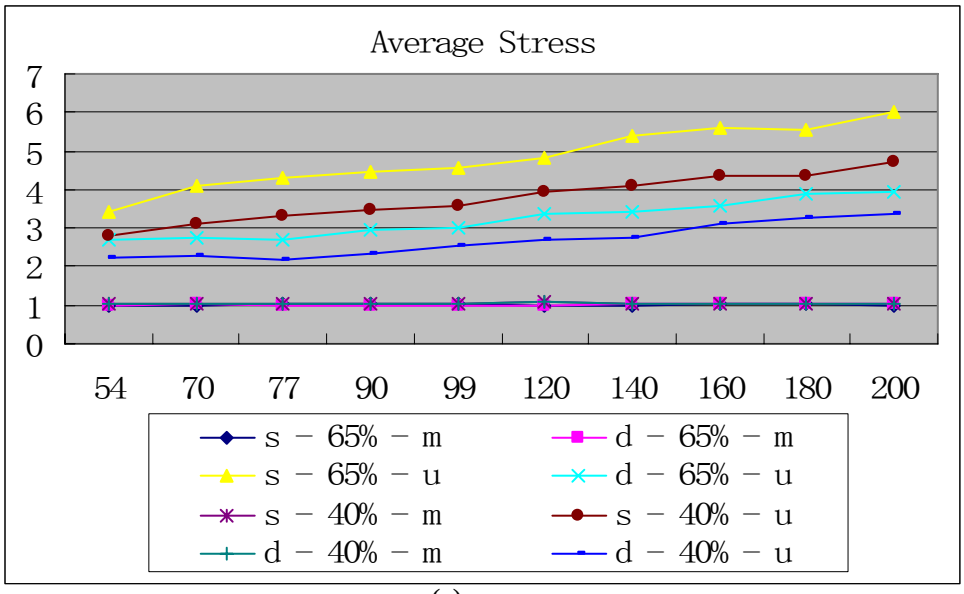
that, compared with unicast, the total transmission cost of multicast is very low, no larger than 50% of the unicast cost. For different network models and run-time parameters, the sparse model with 65% edge routers achieves the lowest ratio while the dense model with 45% edge routers has the highest. This relates to the actual session size, as the sparse model with 65% edge routers possesses the largest session size. However, in the cases of dense model with 65% edge router and sparse model with 40% edge routers, although the former possesses a larger session size than the latter, the ratio is nearly equal.

Therefore, regarding transmission cost, on the whole, the performance of the ALM algorithm in OverMon is close to that of IP multicast, and saves more than 50% transmission cost when compared with overlay unicast. The performance improvement is most obvious when the session size increases; however, by increasing the average node degree does not necessarily achieve better performance.

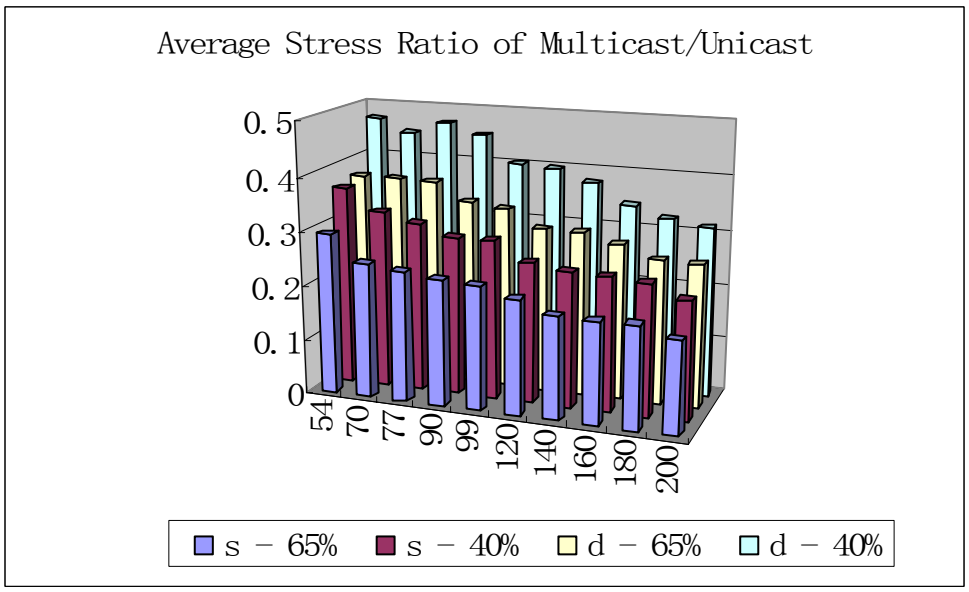
Stress

In Figure 6-14, the evaluation of link stress is presented, with plot (a) showing the average link stress of multicast and unicast in different network models and configurations, and plot (b) showing the ratio of average link stress between multicast and unicast.

As seen in plot (a), the link stress of unicast, for both sparse and dense mode, shows the highest values; when the session size is larger, the link stress is higher. In contrast, the link stress for multicast is steadily around 1.0, which is nearly equal to that of IP multicast. The ratio of link stress between multicast and unicast is plotted in (b). It exhibits that the sparse model with 65% edge routers achieves the best performance improvement, while the dense model with 40% edge routers achieves the worst. For the sparse model with 40% edge routers, although the session size is smaller than the dense model with 65% edge routers, the stress ratio is slightly smaller and better.



(a)



(b)

Figure 6-14: Control overlay evaluation - stress in simulation

Stretch

The evaluation of link stretch is plotted in Figure 6-15; plots a–d represent the results of simulating the sparse and the dense network models with edge routers proportional to 65% and 40%, respectively.

Consistently, the minimum stretch for the four cases is around 1.0, which is very close to that for IP multicast. Plots also show that most of the stretch values are low, leading to

the average stretch consistently around 1.5-2.5; the worst cases of maximum stretch appear in the sparse mode with 65% edge routers - being 4 when the network size is 54 and being12 when the network size reaches to 180. .

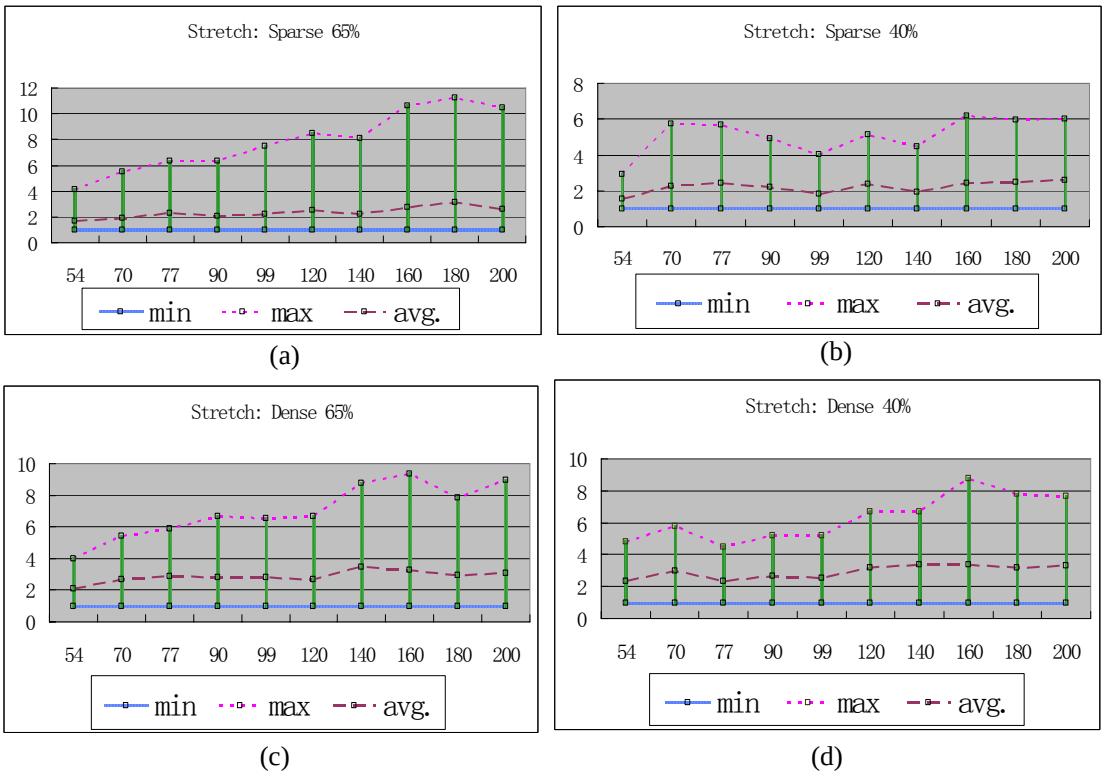


Figure 6-15: Control overlay evaluation – stretch in simuation

For different network models, when the proportion of edge routers is 65%, the dense model achieves better stretch than the sparse model; while when the proportion of edge routers is 40%, the sparse model performs better. This shows that when the proportion of edge router is large, implying that the proportion of nodes participating in multicast sessions is large, increased node degree does help in improving stretch; however this is not applicable when the proportion of edge router is small. For the same model but different proportion of edge routers, both modes show that with fewer edge routers (smaller session size) better stretch can be achieved.

Therefore, as a summary of simulation results, with different network settings, slight differences exist in performance. However, on the whole, the OSPF snooping approach achieves good performance in using bandwidth: regardless of network models, sizes, and the proportion of edge routers, the efficiency in bandwidth usage is close to the level of

IP multicast.

6.5 Evaluation of Data Overlay

This section presents the evaluation of OverMon's data overlay. Due to the fact that OverMon's data overlay is applying Mercury [Bharambe04] as the building block, and extensive simulation has been performed by the original authors, the strategy for OverMon is to conduct evaluation by emulation only, and focus on the performance of topology-aware overlay construction and benchmark testing.

6.5.1 Metrics

With regards to topology-aware overlay construction, its impact on the underlying network layer needs to be evaluated. In Section 6.4.1, three metrics that are widely used to evaluate ALM have been thoroughly discussed, namely transmission cost, link stretch, and link stress. In a data overlay, a single routing/querying operation can not fully reflect the system's performance; transmission cost is used to evaluate data overlay performance, as it gives a better indication, when compared with the other two metrics that are more suitable for evaluating a single multicast session.

In ALM, the transmission cost can be compared with that of IP multicast, and can be normalized by a generalized formula. Unfortunately, in the data overlay, it is difficult to conduct a head-to-head comparison, and a generalized normalization method is even harder, due to the huge variety in functionalities, motivations, and performance requirements etc. Therefore, the metric of transmission cost needs to be re-defined, and the computation method is slightly different:

- **Transmission Cost:** defined as the cost of sending an overlay message from one node to another node, i.e. the cost of an overlay hop; and can be measured by summing up the weight values of an overlay hop along the unicast path. Such a message can be sent either for routing a data item or for a range query; additionally, such messages can be used to maintain neighbour relationships and exchange sampling estimates etc.

Besides transmission cost, to benchmark the performance and overhead of the data overlay in OverMon, the following metrics are used:

- **Range Distribution:** defined as the distribution of the size of the range (i.e. the

value interval) for which a node is responsible; it can be used as a factor to measure the performance of load balancing. Since the data overlay is incrementally constructed, a node's range might change when a new node joins the overlay; thus, this measurement (range distribution) should be made when the system has *stabilised*. For OverMon, the *leave-join* style load balancing is not a research focus, thus this metric is measured after all nodes join the overlay, since from that point on, the range for each node does not change any more.

- **Routing Load Distribution:** defined as the distribution of the number of operations that a node has performed while processing a routing request; these operations include initiating a routing request, forwarding and resolving a routing message etc. It can be measured via logging such operations at run time at each node.
- **Routing Hop Distribution:** defined as the distribution of the number of overlay hops that occurred during the process of routing a set of data item (or range queries). An overlay hop can be measured by tracking and counting an overlay message when it is routed along the overlay network.
- **Maintenance Overhead in Bandwidth**: defined as the average amount of bandwidth that each node consumes in sending out overlay maintenance messages per unit time (note that this does not include the routing overhead). It can be measured either by the number of messages, or by the size of messages (i.e. the total number of bytes). Given that the implementation details of encapsulating a message might introduce bias in the size of messages, using the number of messages is more objective to reflect the algorithm level characteristics. Note that for both cases, this metric is closely related to the system level configuration, i.e. the frequency of sending such maintenance messages.
- **Maintenance Overhead in System Resources**: defined as the average amount of memory consumed by running the OverMon software. Note that such measurements provide synthetic results involving non-algorithm level factors such as software implementation details, hardware capabilities etc.

6.5.2 Methodology

The evaluation of OverMon's data overlay is conducted via emulation, by running the

developed prototype as described in Chapter 5 on the Xen-based evaluation framework.

Unless stated otherwise, the experiments are run for one hub that represents the integer value of the measurement task ID, with the minimum value as 0 while the maximum value as $n * K$. Note that the setting of K will not constrain interpretation of the results, since the data to be inserted, as discussed below, are *randomly* generated within the range of $\{0, n \times K\}$. That means, ideally with balanced data, each node is responsible for a fixed number of K units of the whole range of values, regardless of the network size. In all experiments that are run, $K = 20$, thus the variance in results from different network size can be seen.

The concrete emulation experiments that have been carried out are illustrated in Figure 6-16, and explained as follows:

- Given an emulated network, a configuration file containing a hub's basic information, such as its name, data type, the minimum and the maximum value etc, is sent to each node. With this file, the first node in the system is in charge of the whole range of the attribute, and splits half of its range to the second node that joins the system.This splitting process continues when more nodes join the system.
- A full list of network nodes is also sent to each node. With this list, nodes are queued into an identical order hence they can locate themselves in the list and randomly choose a node that is in front of it in the list as the node to contact.
- Following the order of the node list, nodes join the system one by one. Once all nodes are started, from the Linux machine *kettle*, a script on each node is triggered to run. This script runs an RMI client to inquire the initial range that each node is assigned when it first joins the system. These initial ranges are collected by *kettle* to verify if there is any overlap or faults with regards to the nodes' ranges.
- Once it is verified that all nodes have joined the system successfully, they are triggered to create experimental data. That is, each node generates 10 experimental data tuples of the form: {taskID, timeStamp, taskResult}. Particularly, the attribute of taskID is randomly generated within the range of $\{0, n \times K\}$, and the overlay hub is formed for this attribute.
- With experimental data being generated at each node, they are inserted into the system by being routed to the node whose responsible range covers the attribute's

value of the data. The process of inserting data into system is configured to repeat for M = 10 times, every 30 minutes, to run the experiment long enough herein to capture the average performance of the system. Note that in the data overlay, when a data value is generated, it is just raw; when being inserted into the system, it is assigned a unique ID, hence although a node may insert the same raw data value for several times, each is processed as a unique data item.

- At run time, the logging component at each node logs information about each overlay message it sends out. The information includes the message's type and the corresponding size in bytes, as well as the destination node's IP address and port number.
- When a node receives a message, and if the message is a routing message, it records the routing behaviour. That is, the *hop-count* field, which is encapsulated in the header of a routing message, is retrieved and increased by one; then if the node is the destination of the message, the final hop-count value is logged for evaluation (otherwise the message is forwarded to the next hop).
- Furthermore, every minute, a system level evaluation is carried out, to generate statistics with regards to bandwidth consumption, memory consumption, number of messages etc.
- When an experiment finishes, the remote scripts are firstly triggered to run on each node to calculate the performance of each node, and the results are then collected and processed locally on *kettle* for further evaluation; separate archives of raw trace files for each node ensures the repeatability and verifiability of the evaluation calculations.

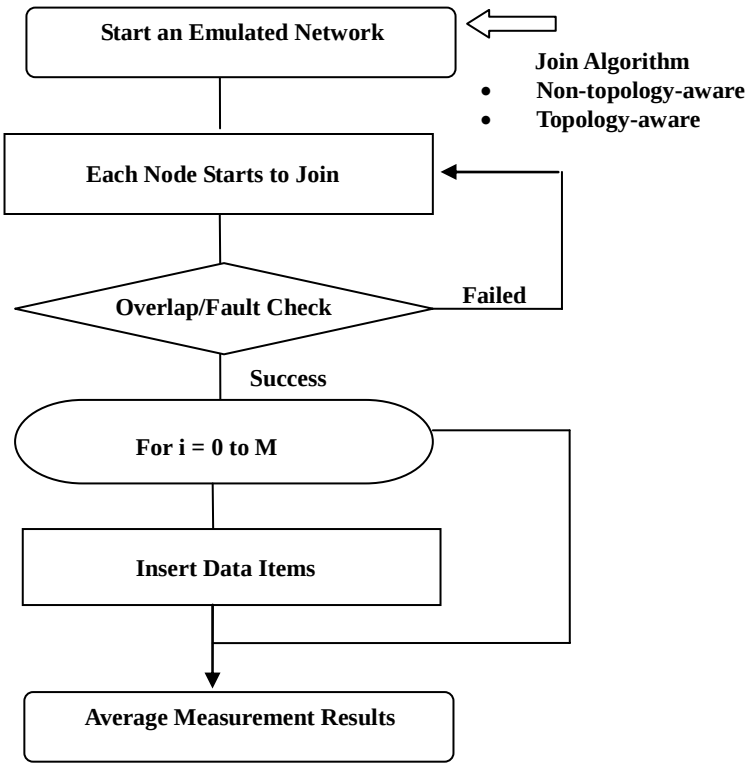


Figure 6-16: Experiment loop of emulation for data overlay

6.5.3 Results

Similar to the control overlay's emulation experiments, the networks used in the emulation experiments follow the sparse model as specified in Table 6-1; and the run-time configurations are largely same as summarised in Table 6-2. Except that, in the data overlay evaluation experiments, due to the limited hardware resources, both core routers and edge routers are used in the experiments. This takes into consideration that there are ten blade servers in total and each is equipped with 2GB memory; with this hardware configuration, the maximum network size that can be achieved is $8 \times 10 = 80$, with each virtual node having 210MB memory and domain 0 having 368MB memory; without loss of generality, when all nodes participate in the experiment, larger networks can be emulated - i.e. the largest network size can be extended from 52 using edge routers only to 77 if both core and edge routers are used.

The results of measuring the metrics as discussed in Section 6.5.1 are shown below, by running the experiment as designed in the previous section for three times and taking the average value. The motivation of running experiment for multiple times is due to the

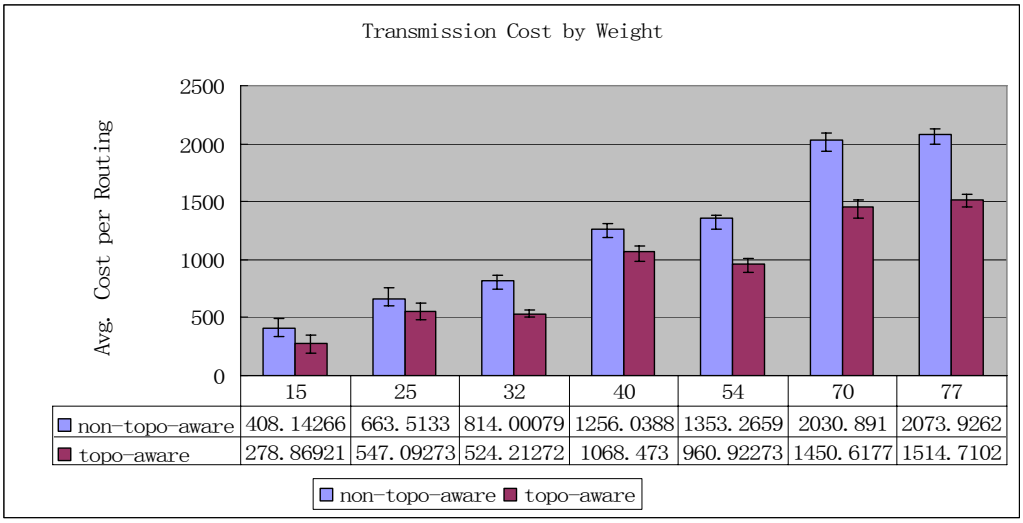
randomness in choosing a node to join and in inserting the data – by running multiple times, it is interesting to see the variance in measurement results. Unless state otherwise, the error bar in the graphs show the value that is above and below the curve, i.e. by calculating the differences of the max value and the min value with the average value of the three experiments. As seen, most of the error bars are small, due to the design of the experiment loop in which M=10 data insertion rounds are performed every 30 minutes, aiming to run the experiment long enough herein to capture the *average* performance of the system.

Transmission Cost

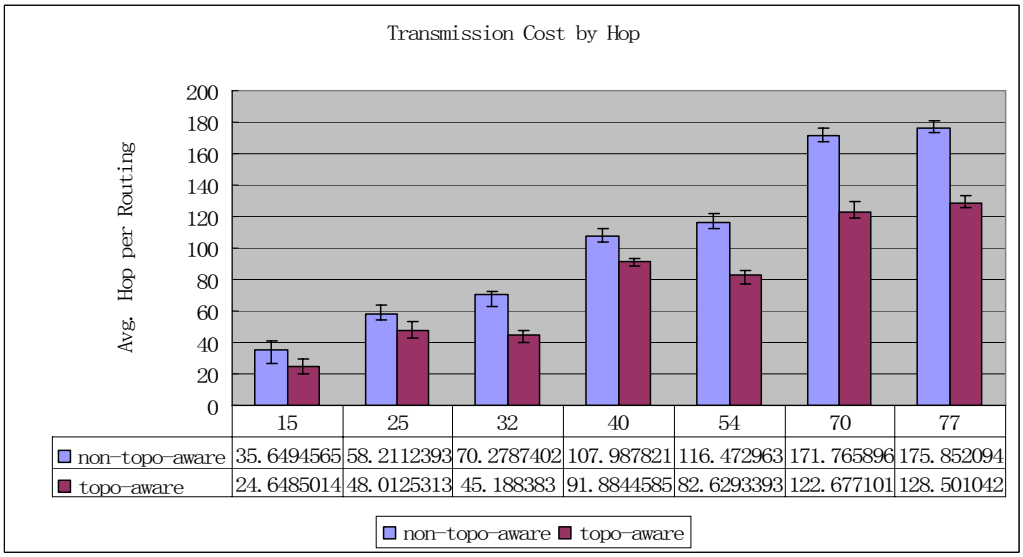
In Figure 6-17, the results of transmission cost of a topology-aware overlay ring vs. a non-topology-aware Mercury ring are plotted, with the y-axis denoting the transmission cost, and the x-axis denotes the network size.

The calculation is as follows: recall that each node logs the messages it sends out, a list of *destination-msgcount* is firstly populated by each sending node; these lists are further aggregated into a *source-destination-msgcount* list. By replacing each *source-destination* pair – i.e. an overlay hop, with corresponding unicast path (i.e. the shortest path along the underlying network level), the transmission cost resulting from this overlay hop can be calculated, either by summing up all weight values along the unicast path, or by counting the hops along the unicast path. By multiplying this cost with *msgcount*, the number of times that a particular overlay hop undertaken in an experiment can be obtained. By summing the resulting value of all the overlay hops, the total transmission cost is then obtained.

Note that, although the experiments are configured to perform a pre-known number of routing tasks, since these tasks are trigged though the RMI interface from *kettle*, uncertain factors, such as the timeout of the *ssh* connection from *kettle to* these virtual nodes, can prevent a successful response to the RMI call, thus a small difference might exist in the number of routing tasks that are triggered successfully. However, the impact of this uncertainty is eliminated by taking the number of successfully triggered tasks into consideration. That is, for fairness, the y-axis depicts the value of dividing the total transmission cost by the number of routing tasks that have been successfully initiated, thus approximately represents the average transmission cost *per routing task*.



(a)



(b)

Figure 6-17: Data overlay evaluation – transmission cost

In Figure 6-17, plot (a) shows the transmission cost calculated by using the weight value of each network link, and plot (b) shows the transmission cost by using the hop number of a network path. As seen, the overlay construction with topology-awareness spends less network resources in terms of both weight value and hop count; the average transmission cost increases linearly with network size growing, due to the larger set of neighbouring links maintenance for each node, and the fact that each node is triggered to insert measurement data one by one, thus there is a short delay (30sec) between each node being triggered causing more maintenance traffic generated.

Range & Routing Distribution

In Figure 6-18, the distributions of nodes' range size and routing load are plotted, for the comparison of a topology-aware overlay ring, vs. a non-topology-aware Mercury ring.

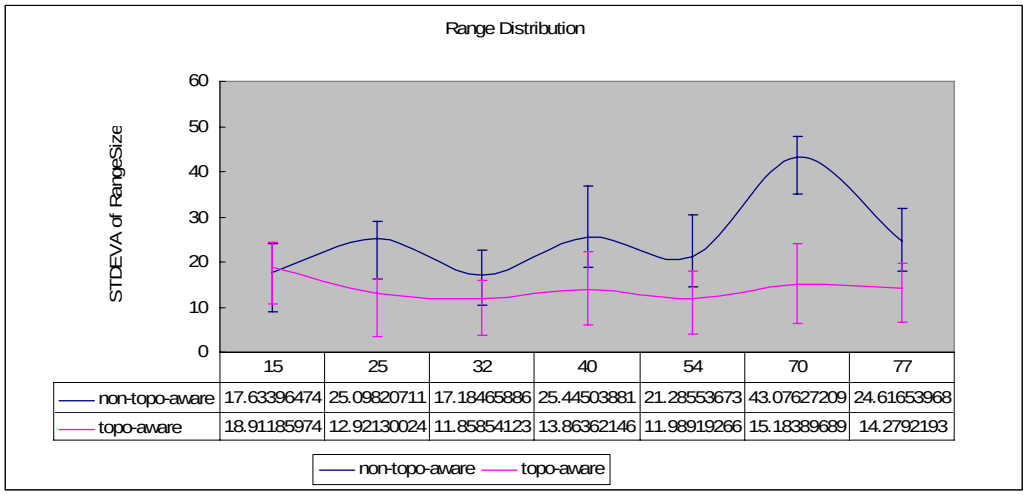
As mentioned, the issue of leave-join style load balancing, i.e. the approach of moving nodes around to adjust range distribution and, therefore, to achieve load balancing, is not extensively studied in OverMon. However, range size balancing is considered in the heuristic overlay construction algorithm, since intuitively, the initial range that each node obtains closely relates to the routing load that a node experiences.

Therefore, the range distribution is measured after all nodes join the system. That is, once the last node joins the system, though an RMI interface, each node is triggered by *kettle* to dump its range information into a log file, then the dumped log files are gathered and aggregated by *kettle*. Note that since a node's range does not change after all node join the system, it could be measured when an experiment finishes. However, the benefit of measuring it at the beginning of an experiment is that any range overlaps or faults can be detected earlier rather than later, thus enabling a decision to be made on whether the experiment should be aborted or not. As to the routing load distribution, it is measured when an experiment finishes, by aggregating each node's logged information for the routing tasks it has performed.

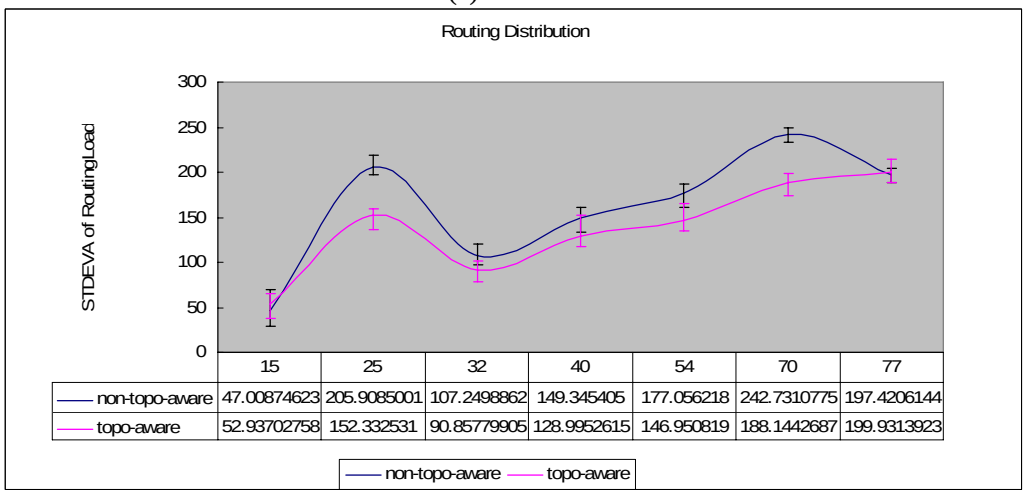
For clearer presentation, the distributions are shown in the form of standard deviation. Taking the range distribution as an example, recall that ideally, regardless of network size, each experiment is designed to make each node responsible for a fixed number of K units of the whole value range, thus the actual size of all nodes' ranges form a set of values, and the spread of these value can be measured by its standard deviation σ; the smaller the σ is, the more even the distribution is. In plot (a), the y-axis denotes σ_r, the standard deviation of the range size for which each node is responsible; in plot (b), the y-axis denotes σ_l, the standard deviation of a node's routing load; in both graphs, the x-axis denotes the varying network sizes.

As seen in plot (a), except network size being 15, the curve for the topology-aware approach is consistently below the curve of the non-topology-aware approach. Similarly, in sub graph (b), the curve of the topology-aware approach is consistently below the curve of the non-topology-aware approach, although the gap is not as big as in plot (a). The common trend in the two graphs shows that, given the same randomness of contacting a node to start the join process, both range size balance and routing balance are achieved better by the heuristic overlay construction algorithm, which makes a

compromise between transmission cost gain and range size balance gain.



(a)



(b)

Figure 6-18: Data overlay evaluation - range & routing distribution

Additionally, the error bars, which stand for the differences of the max value and the min value with the average value of the three experiments, show that the variance in range distribution is relative larger than other measurements, particularly in the case of range distribution in plot (a). This is due to the randomness in choosing a node to join herein the randomness in obtaining an initial range for each node: recall that in current implementation of OverMon, each node's range does not change after the range is assigned; by running three times of the experiment, it is reasonable that the variance in standard deviation of the range size is relative large. However, in plot (b), the error bar shows the variance in routing load is smaller. This is due to the randomness in data insertion and long time lasting of the experiment: recall that for each experiment, every

30 minutes, 10 data items are randomly generated and inserted; and this data insertion is iterated for 10 times, it is reasonable that the standard deviation of routing load for each experiment is an well averaged result across all nodes and the variance for three experiments is small.

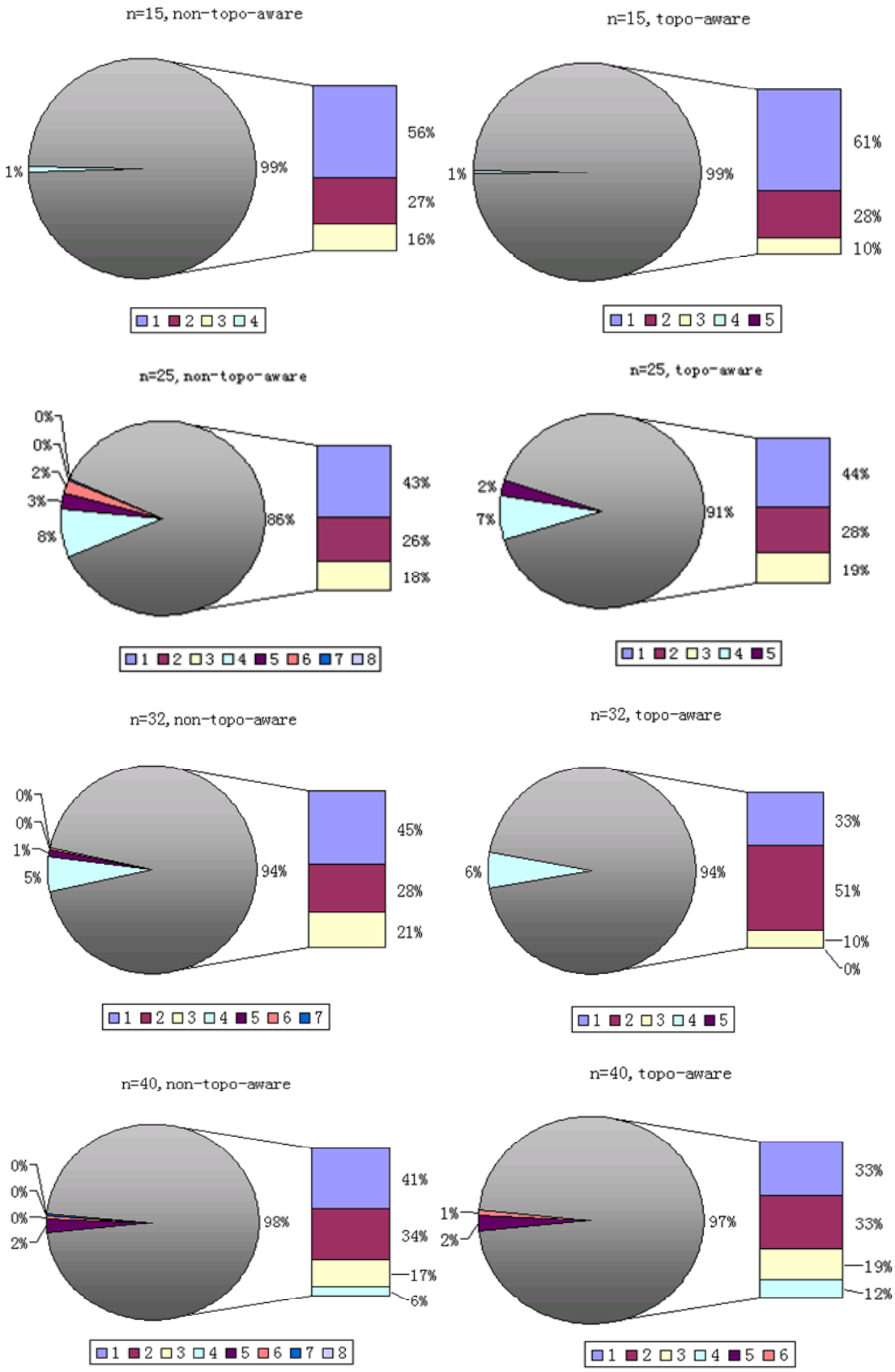
Routing Hop Distribution

Calculated by accumulating the results from three experiments, in Figure 6-19, the routing hop distributions for different network sizes are shown in pie charts, with the left-hand side charts denoting the result for the non-topology-aware approach, and right-hand side charts denoting the result for the topology-aware approach. In addition, the statistics regarding the frequency of shortest hops that occurring in a routing path is summarized in Table 6-4, with the left-hand side columns under a given network size denoting the results of the non-topology-aware approach, and right-hand side columns denoting the results of the topology-aware approach.

As seen from the pie charts, most of routings can be completed in a small number of hops, although the topology-aware approach does not always perform better than the non-topology-aware approach when the hop count is limited to 1, and especially for larger network size such as, 54, 70 and 77. However, starting from a hop count limited to 2, the topology-aware approach performs better than the non-topology-aware approach when the network size is small; and can perform approximately the same or even better for larger network sizes when the hop count is limited to 3 or 4.

In addition, relatively long hop routing does occur when network size grows, and for a given network size, most long hop routings occur in the non-topology-aware approach. The most obvious example is when the network size is 70, the longest routing hop reaches to 12 in the non-topology-aware approach, while just 7 in the topology-aware approach.

Given that the overlay routing is purely at the application level based on the data value, whereas the topology-aware overlay construction takes the network level hops as the primary priority to consider, intuitively the topology-aware approach will not impact the routing hop distribution. Fortunately, since the heuristic algorithm also considers the range size balancing, the resulting overlay is more balanced in terms of both range size and routing load (as seen from Figure 6-18). As a consequence, in the topology-aware data overlay, most routing tasks can be completed in 3 or 4 hops, without very large routing hops occurring.


n=15, non-topo-aware
1%
99%
56%
27%
16%
1 2 3 4
n=15, topo-aware
1%
99%
61%
28%
10%
1 2 3 4 5
n=25, non-topo-aware
0%
0%
2%
3%
8%
86%
43%
26%
18%
1 2 3 4 5 6 7 8
n=25, topo-aware
2%
7%
91%
44%
28%
19%
1 2 3 4 5
n=32, non-topo-aware
0%
0%
1%
5%
94%
45%
28%
21%
1 2 3 4 5 6 7
n=32, topo-aware
6%
94%
33%
51%
10%
0%
1 2 3 4 5
n=40, non-topo-aware
0%
0%
0%
2%
98%
41%
34%
17%
6%
1 2 3 4 5 6 7 8
n=40, topo-aware
1%
2%
97%
33%
33%
19%
12%
1 2 3 4 5 6

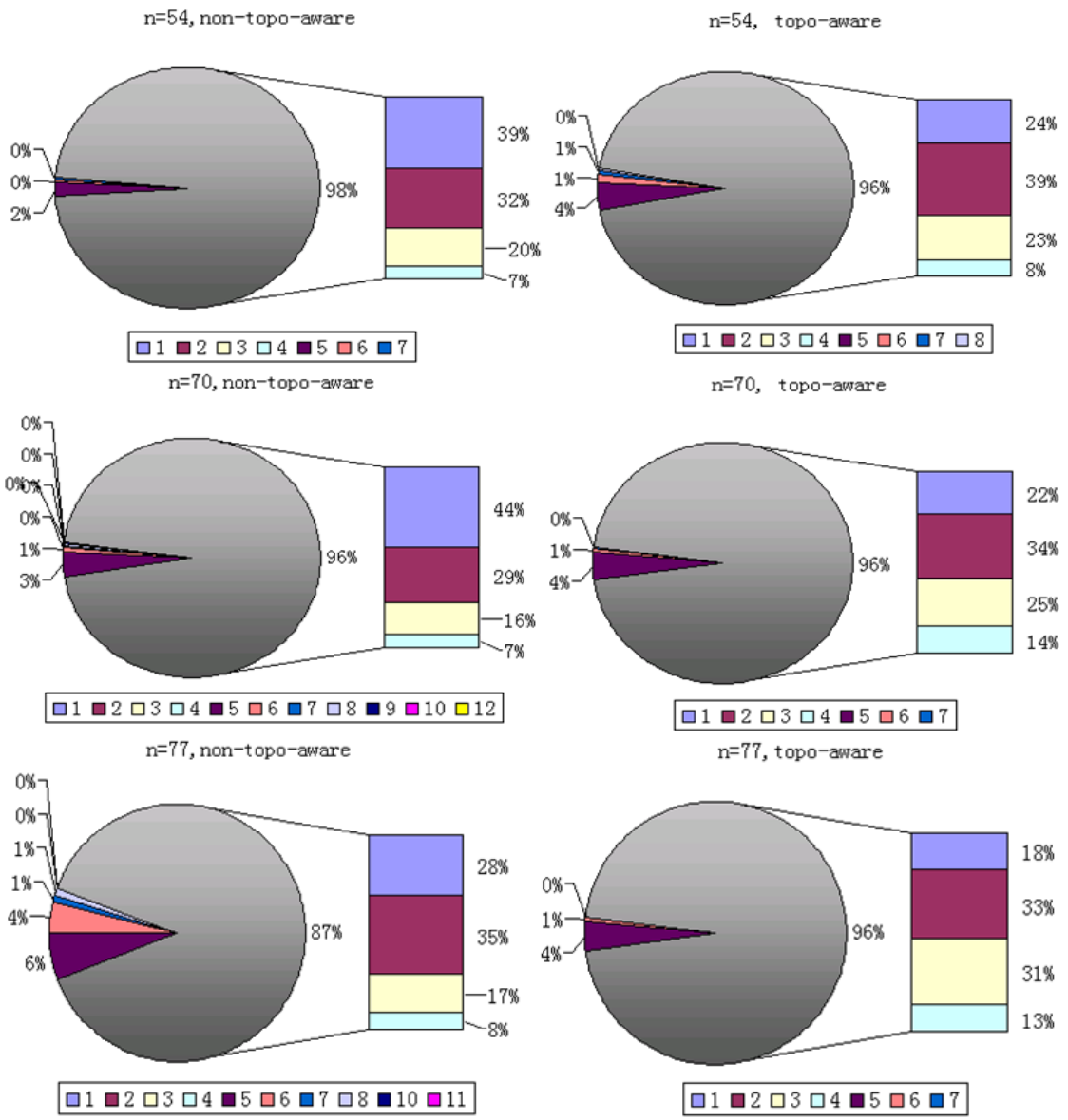


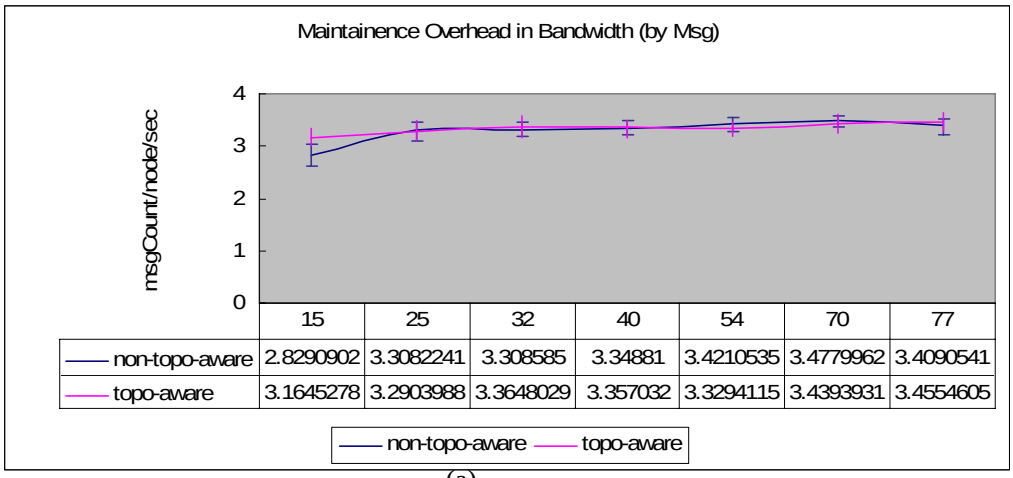
Figure 6-19: Data overlay evaluation - routing hop distribution

Table 6.4: Statistics on the distributions of routing hops in data overlay

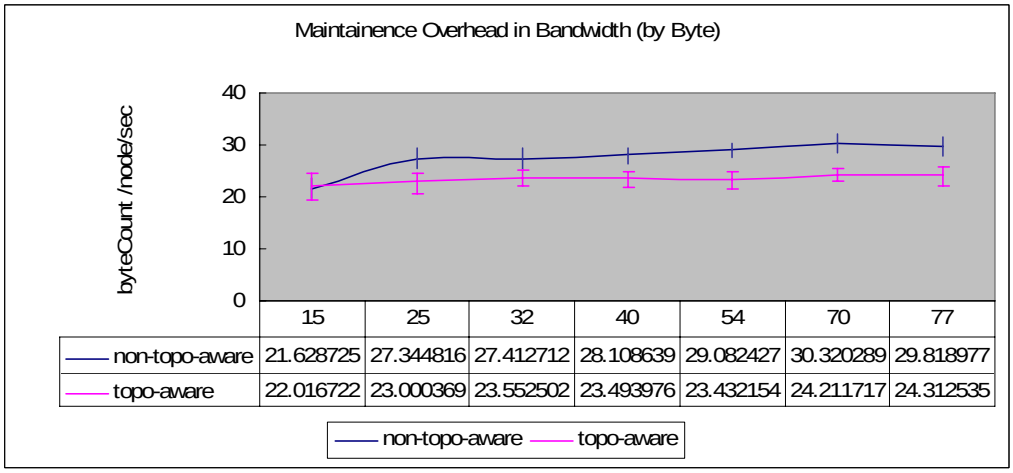
Frequency of Hop	N=15 (%)		N=25 (%)		N=32 (%)		N=40 (%)		N=54 (%)		N=70 (%)		N=77 (%)	
1	56	61	44	43	44	43	40	43	39	24	43	22	28	18
2	83	89	49	71	72	84	73	66	71	62	72	56	63	51
3	99	99	87	90	93	94	91	85	90	86	88	81	81	82
4	n/a	99	95	98	98	100	97	97	98	94	95	96	89	96

Maintenance Overhead in Bandwidth

In Figure 6-20, the average maintenance overhead in bandwidth is plotted, with plot (a) denoting the average number of maintenance messages that each node sends per second, and plot (b) denoting the corresponding size of these messages in bytes.



(a)



(b)

Figure 6-20: Data overlay evaluation - maintenance overheads in bandwidth

The results are calculated on a per-minute basis measurement by each node. That is, for every single minute, each node accumulates the number of messages, as well as the size of messages, that it sends out within this one minute period. After dividing the value by 60 (seconds), this information is appended to a log file as an averaged per-second measurement record for this particular minute. The running time of each node is also logged, thus the total running time in minutes can be calculated. Once an experiment finishes, by summarizing the per-second measurement records within the log file and dividing the summarized value by the number of minutes, the averaged per-second measurement for each node is calculated; then the averaged value for all nodes across the

system can therefore be calculated. By using this fine-grained approach, the accuracy can be improved and the computation on extremely large data sets can be avoided.

As defined, the value for this metric relates closely to the system level configurations, i.e. the frequency of sending maintenance messages. The configuration for the results shown in Figure 6-20 is as follows: both local estimates and random samplings are exchanged every 5 minutes, long-distance links are repaired every 15 minutes, and the histogram of node count is produced every 30 minutes.

As seen, in plot (a), for topology-aware and non-topology-aware approaches, the number of messages sent by each node per second is about 3.34 and 3.98, respectively, and is quite close to each other for most of emulated network sizes (except the smallest network size with 15 nodes). In plot (b), the curves show that for topology-aware and non-topology-aware approaches, the average size of the messages that each node sends per second is about 23.43 bytes and 27.67 bytes respectively; for the topology-aware approach, less maintenance traffic, about 15% on average, is expended. Also, for both graphs, the curves are relatively flat, regardless of the network size. This is due to the fact that at the algorithm level, the experiments are configured with nodes exchanging local estimates within $N_d = 3$ neighbourhoods and sampling $k_1 = \log N$ neighbours randomly; obviously for smaller network sizes like $N = 15,25,32,40,54,70,77$, $k_1 = \log N$ is quite close, thus no sharp differences appear.

Table 6.5: Overhead saving as the result of adjusting maintenance frequency

Estimate Exchange Rate	**Random Sampling Rate**	**Long-Link Repair**	**Histogram Generation**	**Msg Count perNode perSecond**		**Msg Size perNode perSecond**	
				Topo-aware	**Non-topo-aware**	**Topo-aware**	**Non-topo-aware**
5 min	5 min	15 min	30 min	3.46	4.28	24.31	29.82
10 min	10 min	30 min	60 min	1.77	1.89	12.38	13.04
Bandwidth Overhead Saving				51%	44%	51%	44%

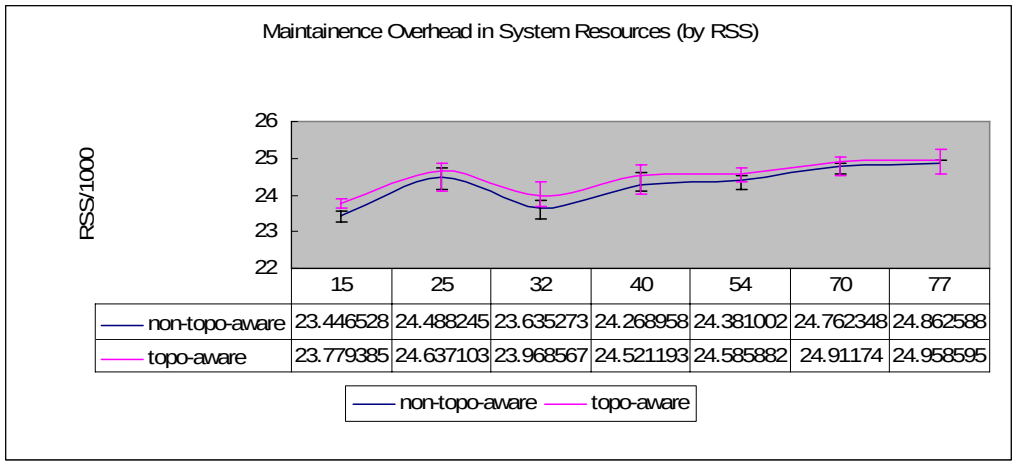
By changing the system level configurations, e.g. by sending maintenance messages less frequently, the overhead for each node is linearly decreased. This obviously is a trade-off

between accuracy and overhead, and can be proved easily by running an experiment on the network consisting of 77 nodes, with the adjusted exchanging rate of local estimates and random samples from 5 minutes to 10 minutes, long-distance link repairing rate from 15 minutes to 30 minutes, and node count histogram constructing rate from 30 minutes to 60 minutes. The results show that by halving the frequency of maintenance messages, the message count and message size both decrease by approximately half, as shown in Table 6-5.

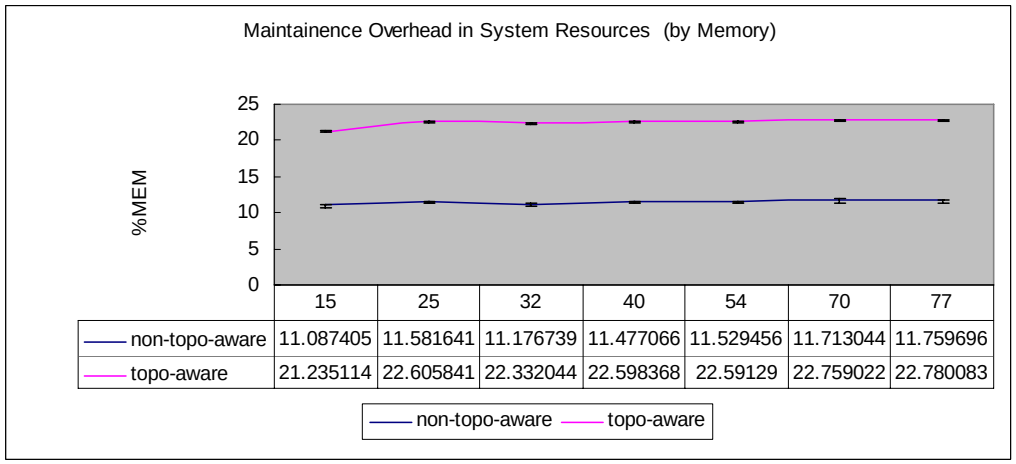
Maintenance Overhead in System Resources

Finally, the maintenance overhead in terms of system level resources are plotted in Figure 6-21, with plot (a) denoting the average RSS (Resident Set Size) usage (i.e. by OverMon processes, the pages of virtual memory that are in RAM), and plot (b) denoting the percentage memory usage. Both are calculated by executing the Linux command "*ps –A v*" on a per-minute basis at each node, and by averaging the logged records from all nodes. Note that for clearer presentation, the y-axis in sub-graph (a) is divided by 1000.

As seen from plot (a), the average RSS usage of the topology-aware approach to the non-topology aware approach is not in sharp difference (the ratio is about 1.009). In plot (b), the difference in averaged percentage memory usage of the topology-aware approach and the non-topology aware approach is large (the radio is about 1.949). The extra memory consumed by the topology aware approach is mainly due to the maintenance of the OSPF topology database. This has been established by running the experiments with the OSPF snooper disabled, with the network topology being obtained through a static topology file; the differences caused by the heuristic overlay construction algorithm are minimal. Although the extra memory usage introduced by the topology-aware approach is considerable, it is reasonable to be assumed that if OverMon can somehow interface with the OSPF daemon, thus to obtain the topology information directly from the OSPF daemon, this extra memory consumption in maintaining a duplicate topology database can be eliminated.



(a)



(b)

Figure 6-21: Data overlay evaluation - maintenance overheads in system resources

In addition, for both approaches, relatively higher memory usage has been introduced, if compared with other system level daemons. Partly, this is due to the implementation details as described in Chapter 5, i.e. the prototype is written in Java, the message encapsulation uses XML-based coding, and a considerable amount of computation overhead is generated by the logging needed for evaluation. Therefore, the scope for optimization is significant. Lastly, given that commercial hardware available in commercial edge routers are much more powerful than the hardware environment of these emulated virtual network nodes, it is reasonable to be optimistic that the gain in bandwidth saving as shown earlier is worth trading for computation resources, such as memory usage.

6.6 Summary

This chapter has presented the systematic evaluation of OverMon.

In Section 6.1, it firstly discussed the general methodologies that are widely accepted to evaluate a large distributed system, and brought forward that the proper strategy to evaluate OverMon efficiently is to treat control plane and data plane differently: for the control plane, the evaluation is based on a combination of simulation and emulation, while for the data plane, the evaluation is based on emulation only.

Given that existing simulation and emulation tools can't be straightforwardly used, a Xen-based emulation toolkit was designed and implemented, as described in Section 6.2. By using this toolkit, emulated networks consisting of core and edge routers running the OSPF protocol can be set up or torn down easily and efficiently, as introduced in Section 6.3.

Next, the construction of application level multicast trees in control plane is evaluated in Section 6.4, using both emulation and simulation. The results show that overlay construction does benefit from topology awareness, especially when the topology information is obtained from pre-existing traffic (OSPF). The performance closely approximates that of IP multicast; and this performance improvement is not significantly affected by different network models in terms of different average node degree and node placement.

The performance of the data overlay is examined in Section 6.5, mainly through emulation. The results show that with topology-aware overlay construction, i.e. by using a heuristic algorithm that makes a compromise between transmission cost gain and range size balancing gain, not only can the transmission cost generated by the data overlay on the underlying network layer be reduced, but, in addition, this improvement does not sacrifice performance at the overlay level in terms of overlay routing hops, balanced range sizes and routing load, and maintenance overhead in bandwidth. The only price that has to be paid is extra memory usage; however, with affordable commercial hardware, this overhead can be tolerated.

Chapter 7

Conclusion

This dissertation proposed a new paradigm for edge-to-edge network monitoring systems through the application of overlay techniques.

The context for this study is continuously evolving network systems, which present new challenges for the network monitoring community, namely, dynamic topologies, rapidly increasing scale and significant complexity. Conventional network monitoring approaches have difficulty in coping with challenges, thus distributed network monitoring has recently been a topic of interest in the network research community. An interesting observation is that even though overlay networks aggravate the complexity and difficulty in managing current and future networks, they provide the required flexibility and scalability – i.e., using overlay networks, new network services and functionalities can be created quickly and easily, and these new changes do not require universal support or adoption from the underlying network layer. OverMon has explored the feasibility of applying overlay techniques to build network monitoring systems.

The remaining chapter summarizes the major contributions first, and then presents directions for future work. Lastly, final remarks are made to complete this dissertation.

7.1 Contributions

OverMon is an edge-to-edge network monitoring system, deployed on edge routers of an autonomous system; active measurement tasks can be initiated from any edge router, and from there, the task configuration parameters can be disseminated along the overlay network to the participating nodes. Furthermore, the measurement results are stored in the distributed repository consisting of OverMon nodes' memories; they are indexed by using a ring-based distributed data structure, which enables the retrieval of measurement

results through range queries.

In short, as a distributed network monitoring system, OverMon achieves scalability by using decentralized overlay networks, in which OverMon nodes form a peer-to-peer relationship, equally sharing the work load and providing the same functionalities; thus, server side bottlenecks and single points of failure that exist in most traditional network monitoring systems are eliminated. OverMon achieves flexibility by providing users the full space of configuration, both in designing an active measurement task, and in retrieving the measurement results; thus, rather than being pre-defined, these functionalities are configured to adapt to users' different needs. Together with careful software design and implementation, particularly the interface design, these features enable OverMon to achieve extensibility, as new services and functionalities can easily be added in without changing the whole software infrastructure.

The detailed design and implementation of OverMon has been shown in Chapter 4 and 5 respectively; the detailed evaluation of bandwidth-efficient overlays has been shown in Chapter 6. The following subsections summarise the findings and conclusions of this dissertation at a finer level of granularity.

Problem Identification

This dissertation identifies a practical subset of network monitoring functionality that requires improved flexibility, scalability and extensibility.

As discussed in Section 2.2.1, in a network management loop, there is a subtle difference between the activities of network measurement and network monitoring; the testing environment that network monitoring provides is the basis upon which a network measurement can be conducted. In this sense, a network monitoring system is more fundamentally important, and needs to be flexible, scalable, and extensible to accommodate different types of measurement activities.

Applicability Analysis

This dissertation critically analyses the application of current overlay network technologies to network monitoring systems.

Given that overlay technologies have attracted tremendous attention from the network and system community, OverMon needs to determine the feasibility of applying overlay technologies to network monitoring systems, and furthermore, to distinguish the applicability of potential techniques. This is done firstly by a comprehensive definition of OverMon from different angles (as discussed in Section 3.2), then a critical analysis of

the applicability of relevant techniques (as discussed in Section 3.3). As a result, application level multicast (ALM) was chosen as the control plane to initiate monitoring tasks, and overlay based data management was chosen to index and retrieve measurement results.

However, the primary issue with overlay networks is the significant network overheads required to maintain the overlays. To make OverMon practically useful, this issue had to be addressed.

Design and Implementation

Architecturally and algorithmically, this dissertation described bandwidth-efficient overlay techniques for use in network monitoring, to cope with the network overheads of overlay networks.

By snooping on interior gateway protocol (IGP) traffic, e.g. OSPF traffic, among the routers in a network, the network topology can be reconstructed (as discussed in Section 4.2), which is important in maintaining and optimizing overlay structures; in normal overlay networks, the network topology can only be approximated by periodic pinging, leading to incomplete knowledge of the topology and contributing to network overheads.

By exploiting such IGP information to construct application-level multicast trees among edge routers of the network to facilitate measurement control traffic, (i.e. the traffic for measurement task configuration), the results, as discussed in Section 6.4.3, show that overlay construction does benefit from topology awareness, especially if the topology information is obtained from pre-existing traffic (OSPF). The performance of OverMon closely approximates that of IP multicast; and this performance improvement is not significantly impacted by different network models in terms of different average node degree and node placement.

By exploiting such IGP information to construct application-level ring structures among edge routers of the network to facilitate access to accumulated measurement data, the results, as discussed in Section 6.5.3, show that with topology-aware overlay construction, i.e. by using a heuristic algorithm (as discussed in Section 4.4.3.2) that makes a compromise between transmission cost gain and range size balancing gain, not only can the transmission cost generated by the data overlay on the underlying network layer be reduced, but, additionally, this improvement does not sacrifice performance at the overlay level in terms of overlay routing hops, balanced range sizes and routing load, and maintenance overhead in bandwidth. The only price that has to be paid is extra memory usage; however, with affordable commercial hardware, this overhead can be tolerated.

All these techniques and approaches have been implemented in a Java-base prototype, as discussed in Chapter 5. The implemented software maximizes the flexibility and extensibility of OverMon, without sacrificing its performance.

Evaluation Methodology

This dissertation provides an experimental system design and performance study of these bandwidth-efficient overlay techniques applied to network monitoring to demonstrate the efficacy of the approach. Of particular interest is the design of a Xen-based emulation framework, and a customised simulation solution for the experimentation, as discussed in Section 6.2.2 and Section 6.4.2.2 respectively.

Given that most of the existing work has resorted to simulations to evaluate the effectiveness of proposed solutions, and that a fully controlled simulation environment is decoupled from any external traffic or system, simplified assumptions may result in inaccurate representations of network dynamics as seen in real-world environments. In contrast, the Xen-based framework can maximize the approximation of the emulated environment to a real testbed, since a large number of virtual nodes can be federated to act as edge routers running real protocols, the network topologies that are under test are generated by a well-accepted topology generator, and the modules under test are real implementations interacting with the protocol stack of the underling emulator host. Within this realistic environment, OverMon can be reliably evaluated.

In addition, as a complementary means, a customised simulation solution catering for OverMon’s control plane has been implemented. With this simulation software, by comparing the results of running simulation and emulation on identical networks, the two sets of results are mutually verified: the simulation software is correctly implemented, and the network topology formed by snooping from OSPF packets is correct. Then, larger network sizes that exceed the emulation capacity, or different network models, can be simulated with confidence in the simulation results due to the validation against the emulation experiments.

7.2 Future Work

There are several areas of future work that could be pursued.

Network Measurement & Monitoring

One direction is to accommodate additional network measurement techniques, such as

supporting passive measurement. In this case, a passive measurement task involving one or more measurement sites can be initiated in the same way as an active measurement task in OverMon, i.e. though application level multicast. However, regarding the processing of measurement results, although the flexibility provided by range query may still hold, other data management techniques might be more suitable, e.g. continuous aggregation of passive measurement results. Thus, it would be interesting to explore the feasibility of co-existence of continuous aggregation and on-demand range query in OverMon.

Given the more comprehensive view of network performance status that results from support of additional measurement techniques, another direction is the possible application of advanced data processing techniques, as discussed in Section 2.2.2.3, to facilitate additional network management functionalities. For example, an integrated closed-loop network management system could be formed, in which the measurement sites required to participate in a measurement can be inferred and determined by the system itself, rather than being specified by users.

Overlay Construction & Maintenance

In OverMon, the network topology information is leveraged to build two overlay networks; both have achieved performance improvement in terms of being bandwidth efferent. However, the degree of improvement is different. For the control overlay, where the multicast tree is constructed largely relying on the metric of distance, i.e. the total minimum distance of the tree at the overlay level, the resulting performance is nearly as optimal as IP multicast. For the data overlay, where the overlay path is constructed largely using an application level metric, i.e. the range value, the usage of topology information is more limited, only at the stage of choosing the bootstrapping node. Hence it would be interesting to investigate to what degree the topology structure of an overlay network impacts on its performance improvement, particularly with an accurate network topology being available, and if the overlay is to support both continuous aggregation and on-demand range query.

Implementation and Evaluation

From an implementation perspective, currently, OverMon maintains its own topology database by snooping OSPF packets. Although this method is straightforward to implement and does not produce any network overhead, it enforces extra usage of system resources (i.e. CPU cycles and memory) on edge routers, since duplicate topology databases (by the OSPF daemon and by OverMon) are maintained on each edge router. It

would be interesting to investigate if this duplication can be removed, with OverMon accessing the needed topology information directly from the OSPF daemon.

Finally, the Xen-based emulation framework has proved to be efficient in evaluating a distributed system like OverMon. However, given that OverMon is not a delay-sensitive application, the framework so far is mainly used for the measurement of non-delay sensitive metrics, such as bandwidth consumption, message count, hop count etc. It would be interesting to investigate whether this framework can be extended into a more generalized emulation solution, for example taking timing and dynamic changes in network condition into account; or at least the impacts caused by the limits of Xen virtualisation can be quantified.

7.3 Final Remarks

As current and next generation networks continue to grow in scale and complexity, a shift in the architecture of monitoring systems from a centralized paradigm to a distributed paradigm is needed; the objective is to enable the monitoring tasks being performed in a flexible, scalable, and extensible style.

The overlay network approach is one promising strategy for this goal, and this work takes a first step in this direction. Particularly, this study was motivated to address realistic scale and complexity problems, and the resulting system has been evaluated with realistic implementation and environmental settings. The first hand experiences accumulated from this work no doubt laid the foundation for extensive and deep research work being carried out in the future.

Appendix

A. Data Structures for AM

Recall that to perform an active measurement with OverMon, users need to specify the measurement metric and the probe structure in the API function. Below are the data structures that are used.

Table A.1: The Java class for performance metric and methodology

```
public class IPPMMetric implements Serializable{

    private byte    metric_name;    // 0:delay, 1:loss, 2:jitter, 3:throughput
    private byte    metric_method;  // 0:singleton, 1:sampling, 2:statistics
    private int     para1 ;
    private int     para2 ;         // refer to relevant technical report for detailed usage
    private long    para3 ;         // information for para1, para2, para3
}
```

Table A.2: The Java class for probe structure

```
public class ProbeStructure implements Serializable{

    private int                 probe_seq;
    private byte                encrypt_mode;
    private ProbeDptTime        dpt_time;     // defined in Table 5.4
    private ProbePackets        pkt_size;     // defined in Table 5.5
    private ProbeEndHosts       end_hosts;    // defined in Table 5.6
}
```

Table A.3: The Java class for probe departure time

```
public class ProbeDptTime implements Serializable{

    private byte        type;
    private Date        start_time;
    private Date        end_time;
    private int         para;
}
```

Table A.4: The Java class for probe packet

```
public class ProbePackets implements Serializable{

    private int         padding_size;
    private int         no_of_packets;
}
```

Table A.5: The Java class for probe sender and receiver

```
public class ProbeEndHosts implements Serializable{

    private Vector<String>      endhosts;
    // each element is in the format of "senderIP-receiverIP"
}
```

Table A.6: The Java class for measurement result

```
public class IPPMeasurementResult implements Serializable {

    private int             taskID;
    private int             nodeID;
    private HashMap         value_tuple; // for singleton method, the size is 1, otherwise >1
    private TypeP           typeP;       // defined in ref RFC 2679 3.8.1
}
```

Table A.7: The Java class for monitoring result

```
public class MonitoringTaskResult implements Serializable {

    private int             success_flag;   // success(0) error (error_code>0)
    private String          taskeID;        // the unique id identifying the registered task

}
```

B. Overlay Inter-node Operations

Table B.1: Inter-node operations for control overlay construction

Role	Operation
Root Node	processRMI_NewTask(originalRMITask) sendMulticast(whoAmI, receiverList, topology, isRoot=true, null) recvMulticast_ACK(almPacket) timeOutCheck(sessionInfo) rmiCallBack(originalRMITask, result)
Internal Node	recvMulticast (almPacket) sendMulticast(whoAmI, treeMap, isRoot=false, received_almPacket) recvMulticast_ACK(almPacket) timeOutCheck(sessionInfo) sendMulticast_ACK(senderID, receiverList, topology, isRoot, packet)
Leaf Node	recvMulticast (almPacket) sendMulticast_ACK(senderID, receiverList, topology, isRoot, packet)

Table B.2: Inter-node operations for data overlay construction – node joining

Roles	Operation

NewComing Node (newNode_A)	send_Hub_Init_Request(bsNode_B, hubInitRequestMsg)
	recv_Hub_Init_Reply(hubInitReplyMsg) // from bsNode_B
	send_BootStrap_Request(bsNode_B, bsRequestMsg)
	recv_BootStap_Reply(bsReplyMsg) // from bsNode_B
	send_Successor_Notify(bs's_prodNode_D, succNotifyMsg)
	send_crossHubRequest(crossHub_R ,crossHubLinkRequestMsg)
	recv_crossHubReply(crossHubLinkReplyMsg) //from crossHub_R

BootStrap Node (bsNode_B)	recv_Hub_Init_Request(hubInitRequestMsg) // form newNode_A
	send_Hub_Init_Reply(newNode_A,hubInitReplyMsg)
	recv_BootStrap_Request(bsRequestMsg) // form newNode_A,send_BootStrap_Reply(newNode_A, bsReplyMsg)
	send_Range_Changed(bs's_succNode_C, rangeChangedMsg)

BootStrap Node's Successor
(bs's_succNode_C)
	recv_Range_Changed (rangeChangedMsg) // form bsNode_B,
	// still takes bsNode_B as predecessor but with updated halved range.
	update_Prodecessor(bsNode_B) // reset bsNode_B with updated halved range.

BootStrap Node's Predecessor (bs's_prodNode_D)	recv_ Successor_Notify (succNotifyMsg) // form newNode_A set_ex_Successor (bsNode_B) // reset bsNode_B from successor to ex-successor, with updated halved range.

Cross-Hub Represent Node
(crossHub_R)
	recv_crossHubRequest(crossHubLinkRequestMsg) //from newNode_A
	send_crossHubReply(newNode_A,crossHubLinkReplyMsg)

Table B.3: Inter-node operations for data overlay construction – node leaving

Roles	Operation
LeavingNode (leaving_A)	send_LeaveNotify(succ_B leaveNotifyMsg) send_LeaveNotify(prod_D, leaveNotifyMsg) send_LinkBreak(long_L, linkBreakMsg) send_LinkBreak(cross_R, linkBreakMsg)
Leaving Node's Successor (succ_B)	recv_LeaveNotify (leaveNotifyMsg) // form leaving_A send_Range_Changed (succ_B's Successor, rangeChangedMsg)
Leaving Node's Predecessor (prod_D)	recv_ LeaveNotify (leaveNotifyMsg) // form leavingNode_A send_Successor_Notify(prod_D's Predecessor, succNotifyMsg) //to notify its predecessor the new range
Long-Neighbour Links (long_L)	recv_LinkBreakMsg(linkBreakMsg) // form leavingNode_A
Cross-Hub Represents (cross_R)	recv_LinkBreakMsg(linkBreakMsg) // form leavingNode_A

Table B.4: Inter-node operations for data overlay construction – long neighbour

Roles	Operation

Requesting Node
(req_A)
	sendLongNeighborRequest(valueV, relaying_B, longNbrRequestMsg)
	// relaying_B is calculated basing on the *valueV*, by using the harmonic distribution
	receiveLongeighborReply(LongNeighborReplyMsg)// from reply_C
	set_LongNbr (reply_C)

Relaying Node (relaying_B)	receiveLongeighborRequest(longNbrRequestMsg) sendLongNeighborRequest(valueV, reply_C, longNbrRequestMsg) // relayingNodeB passes this message down to the next hop (e.g. reply_C) if its range does not cover the *valueV*,
Responding Node (reply_C)	receiveLongeighborRequest(longNbrRequestMsg) send_LongNeighborReply(req_A, LongNbrReplyMsg) //if reply_C's range covers the *valueV* set_LongNbr (req_A)

Table B.5: Inter-node operations for data overlay - random sampling

Roles	Operation

PeeringNodeA	send_LocalEstimateRequest(peeringNodeB, LocalEstimateRequestMsg)
	recv_LocalEstimateReply (LocalEstimateReplyMsg)
	// from peeringNodeB
	send_RndSamplingRequest(peeringNodeB, RndSamplingRequestMsg)
	recv_RndSamplingReply (RndSamplingReplyMsg)
	// from peeringNodeB

Table B.6: Inter-node operations for data overlay - routing

Roles	Operation

PeeringNodeA	route (routingMsg, from)// data item or queries, to peeringNodeB
	recv_routing(routingMsg, from)

Bibliography

[Albrecht03] A. Albrecht, R. Arnold, M. Gahwiler, and R.Watterhofer. “Join and Leave in Peer-to-Peer systems: the DASIS approach”, Technical Report, ETH, Zurich, 2003.

[Andrzejak02] Artur Andrzejak Zhichen Xu “Scalable, Efficient Range Queries in Grid Information Services”, in Proc. of the Second International Conference on Peer-to-Peer Computing, 2002

[Anwitaman05] Anwitaman Datta, Hauswirth, M., John, R., Schmidt, R., Aberer, K. “Range queries in trie-structured overlays”, in Proc. of Fifth IEEE International Conference on Peer-to-Peer, 2005.

[Aspnes03] J. Aspnes and G. Shah “Skip graphs”, in Proc. of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, 2003.

[Ayalvadi03] Ayalvadi J. Ganesh, Anne-Marie Kermarrec and Laurent Massoulie “Peer-to-Peer Membership Management for Gossip-Based Protocols”, IEEE Transactions on Computers, VOL. 52, No. 2, February 2003

[Ballardie93] Ballardie, A.J., Francis, P.F. and Crowcroft, J., "Core based trees", in Proc. of ACM SIGCOM, San Francisco, pp, 85-95 1993

[Banerjee02] Banerjee, C. Kommareddy, and B. Bhattacharjee “Scalable application layer multicast”, in Proc. of ACM SIGCOMM, Aug. 2002.

[Barabasi99] A. L. Barabasi and R. Albert “Emergence of scaling in random networks”. Science, 286(5439):509–512, 1999.

[Barham03] Paul Barham, Boris Dragovic, Keir Fraser, “Xen and the Art of Virtualization“, in Proc. of the nineteenth ACM symposium on Operating systems principles, 2003

[Bharambe04] Ashwin R Bharambe, Mukesh Agrawal, and Srinivasan Seshan "Mercury: Supporting Scalable Multi-Attribute Range" , in Proc. of SIGCOMM , 2004

[Binzenhöfer06] Andreas Binzenhöfer, Kurt Tutschku, Björn auf dem Graben, Markus Fiedler and Patrik Arlos “A P2P-based Framework for Distributed Network Management”, New Trends in Network Architectures and Services, LNCS 3883, Loveno di Menaggio, Como, Italy, 2006.

[Birrer04] S. Birrer, D. Lu, F. E. Bustamante, Y. Qiao, and P. A. Dinda. “Fatnemo: Building a resilient multi-source multicast fat-tree”, In WCW, volume 3293 of Lecture Notes in Computer Science, pages 182–196. Springer, 2004.

[BitTorrentWeb] BitTorrent - http://bitconjurer.org/BitTorrent/

[BRITEWeb] http://www.cs.bu.edu/brite.

[Byers03] John Byers, Jeffrey Considine, and Michael Mitzenmacher, “Simple Load Balancing for Distributed Hash Tables” in Proc. of IPTPS, Feb. 2003.

[Chalmers01] Chalmers, R.C., Almeroth, K.C. “Modeling the Branching Characteristics and Efficiency Gains in Global Multicast Trees ” in Proc. of IEEE INFOCOM, page 449 -458, April 2001.

[Chawathe05] Y. Chawathe and S. Ramabhadran and S. Ratnasamy and A. LaMarca and S. Shenker and J. Hellerstein “A Case Study in Building Layered DHT Applications”, in Proc. of SIGCOMM, August 2005.

[Cheriton76] Cheriton, D., Tarjan, R.E. “Finding minimal spanning tree”, SIAM J. Comput. 5, 724-742 (1976).

[Chu00] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan and Hui Zhang "A Case for End System Multicast", in Proc. of SIGMETRICS 2000, International Conference on Measurements and Modeling of Computer Systems, June 18-21, 2000

[Chuang98] J. Chuang and M. Sirbu “Pricing Multicast Communications: A Cost-Based Approach”, in Proc. of Internet Society INET, July, 1998.

[CiscoIPM06] http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ipmulti.pdf

[CiscoMng] http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/nmbasics.pdf

[CORBAWeb] http://www.corba.org/

[Cranor03] Chuck Cranor, Theodore Johnson, Oliver Spataschek,” Gigascope: a stream database for network applications”, in Proc. of International Conference on Management of Data, 2003

[Crovella06] Mark Crovella, Balachander Krishnamurthy "Internet Measurement: Infrastructure, Traffic and Applications" ISBN: 978-0-470-01461-5

[Cugola02] Cugola, G.; Picco, G.P. “Peer-to-Peer for Collaborative Applications”, in Proc. of 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW '02) , Vienna, Austria, July 02 - 05, 2002.

[CZhang04] C. Zhang, A. Krishnamurthy, and R. Y. Wang “SkipIndex: Towards a Scalable Peer-to-Peer Index Service for High Dimensional Data”, In Technical Report.

[Dabek04] F. Dabek, R. Cox, F. Kaashoek, and R. Morris “Vivaldi: A Decentralized Network Coordinate System”, in Proc. of SIGCOMM 2004, Portland, Oregon, USA August 30 – September 3

[David01] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, Robert Morris "Resilient Overlay Networks", in Proc. of 18th ACM SOSP, Banff, Canada, October 2001.

[Deering96] Deering. S., D.L., Farinacci, D., Van Jacobson, Liu, C. and Wei, L., "The PIM architecture for wide=area multicast routing", IEEE/ACM Transactions on Networking, 4(2), 153-162, 1996

[Dijkstra59] Dijkstra, E.N. "A note on two problems in connection with graphs", Numer. Math. 1, 269-271 (1959)

[eDonkeyWeb] eDonkey - http://www.edonkey2000.com/

[El-Sayed03] A. El-Sayed and V. Roca and L. Mathy "A Survey of Proposals for an Alternative Group Communication Service", IEEE Network, Vol 17, No 1, pp46-51, January 2003.

[Ericsson02] M. Ericsson, M.G.C. Resende and P.M. Pardalos, "A genetic algorithm for the weight setting problem in OSPF routing", J. of Combinatorial Optimization", vol. 6, pp. 299-333, 2002

[Faloutsos99] M. Faloutsos, P. Faloutsos, and C. Faloutsos "On power-law relationships of the Internet topology", in Proc. of ACM SIGCOMM, pages 251–262, Cambridge, MA, 1999.

[Floyd62] Floyd, R.W. "Algorithm 97: shortest path", CACM 5, 345 (1962)

[Fraleigh01] Fraleigh, C., Diot, C., Lyles, B., Moon, S., Owezarski, P., Papagiannaki, D., Tobagi, F. "Design and Deployment of a Passive Monitoring Infrastructure", in Proc. of Passive and Active Measurement Workshop (PAM2001), Amsterdam, NL, April 23-24 2001

[Francis00] P. Francis. "Yoid: Extending the internet multicast architecture", April 2000.

[Ganeshan04] Ganeshan, P., Bawa, M., and Garcia-Molina, H. "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems", In Conference on Very Large Databases (VLDB) (2004).

[Garey79] M. R. Garey and D. S. Johnson. "Computers and Intractability: A Guide to the Theory of NP-Completeness". San Francisco, W. H. Freeman, 1979.

[Gilbert68] E. N. Gilbert and H. O. Pollak. "Steiner Minimal Trees" SIAM Journal on Applied Mathematics, 16(1):1-29, 1968.

[GNPWeb] http://www.cs.rice.edu/~eugeneng/research/gnp/

[GnutellaWeb] http://www.gnutella.com/

[Godfrey04] Brighten Godfrey, Karthik Lakshminarayanan Sonesh Surana Richard Karp

Ion Stoica, “Load Balancing in Dynamic Structured P2P Systems”, in Proc. of INFOCOM 2004

[Goldszmidt95] G. Goldszmidt Y. Yemini “Distributed Management by Delegation”, in Proc. of the 15th International Conference on Distributed Computing Systems (ICDCS'95)

[Gribble01] Steven Gribble, Alon Halevy, Zachary Ives, Maya Rodrig, Dan Suciu, “What Can Databases Do for Peer-to-Peer” In Proc. Of Fourth International Workshop on the Web and Databases (WebDB) 2001.

[Habib04] Ahsan Habib, Maleq Khan, Bharat Bhargava, "Edge-to-Edge Measurement-based Distributed Network Monitoring", Computer Networks, Vol. 44, Issue 2, Pages 211-233, Feb 2004.

[Halevy04] Halevy, A.Y.; Ives, Z.G.; Jayant Madhavan; Mork, P.; Suciu, D.; Tatarinov, I. “The Piazza peer data management system”, Transactions on Knowledge and Data Engineering, Volume 16, Issue 7, July 2004 Page(s): 787 - 798

[Hegering99] Hegering, H-G., Abeck, S., Neumair, B., “Integrated Management of Networked Systems: Concepts, Architectures, and Their Operational Application”, Year of Publication: 1999 ISBN:1-55860-571-1

[Hoory06] Shilomo Hoory, Nathan Linial, Avi Wigderson, “Expander Graphs and Their Applications”,MERICAN MATHEMATICAL SOCIETY Volume 43, Number 4, October 2006, Pages 439–561 S 0273-0979(06)01126-8

[Huebsch03] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, Ion Stoica “Querying the Internet with PIER”, in Proc. of 19th International Conference on Very Large Databases (VLDB), 2003.

[IDMapWeb] http://idmaps.eecs.umich.edu/

[Jannotti00] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O’Toole, Jr., “Overcast: Reliable multicasting with an overlay network”, in Proc. of the 4th conference on Symposium on Operating System Design & Implementation - Volume 4

[Karger04] Karger, D., and Ruhl, M. “Simple efficient load-balancing algorithms for peer-to-peer systems”. in Proc. of Third International Workshop on Peer-to-Peer Systems (2004).

[Kleinberg00] J. Kleinberg. “The small-world phenomenon: An algorithmic perspective” in Proc. of 32nd ACM Symposium on Theory of Computing, 2000. Also appears as Cornell Computer Science Technical Report 99-1776

[Kortuem01] “When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobil Ad hoc Networks”, Proceedings First International Conference on Peer-to-Peer Computing, Lingköping Sweden, August 2001.

[Kou81] L. Kou, George Markowsky, Leonard Berman," A fast algorithm for Steiner trees" Journal Title: Acta Informatica Date: 1981 Volume: 15p. 141 - 145"

[Kühne06] Mirjam Kühne Nurani Nimpuno Sabrina Wilmot, "Autonomous System (AS) Number Assignment Policies and Procedures", Document ID: ripe-389, Septermber 2006

[Kwon02] M. Kwon and S. Fahmy, "Topology-Aware Overlay Networks for Group Communication" in Proc. of ACM NOSSDAV, May, 2002.

[Lao07] Li Lao,Jun-Hong Cui, Mario Gerla, Shigang Chen, "A Scalable Overlay Multicast Architecture for Large-Scale Applications" IEEE Transactions on Parallel and Dsitributed Systems, April 2007 (Vol. 18, No. 4) pp. 449-459

[Lawder00] J. K. Lawder and P. J. H. King "Using Space-Filling Curves for Multi-dimensional Indexing", Lecture Notes in Computer Science,Volume 1832/2000

[Li04] Baochun Li, Jiang Guo, Mea Wang: iOverlay: A Lightweight Middleware Infrastructure for Overlay Application Implementations. Middleware 2004: 135-154

[Liebeherr99] J. Liebeherr and T. K. Beam. "Hypercast: A protocol for maintaining multicast group members in a logical hypercubetopology", in Proc. of 1st International Workshop on Networked Group Communication (NGC '99), pages72–89, July 1999.

[Liotta02] Liotta, A. Pavlou, G. Knight, G. "Exploiting Agent Mobility for Large Scale Network Monitoring", IEEE Network, special issue on Applicability of Mobile Agents to Telecommunications, Vol. 16, No. 3, IEEE, May/June 2002.

[Lowekamp04] Bruce Lowekamp, Brian Tierney, Les Cottrell, Richard Hughes-Jones "A hierarchy of Network Performance Characteristics for Grid Applications and Services", in Proc. of Network Measurements Working Group, 2004.

[Manku03] G. S. Manku, M. Bawa, and P. Raghavan. "Symphony: Distributed hashing in a small world", in Proc. of 4th USENIX Symposium on Internet Technologies and Systems (USITS 2003)

[MARSWeb] http://www.compu-art.de/mars_nwe/.

[Maymounkov02] P. Maymounkov and D. Mazières "Kademlia: A peer-to-peer information system based on the XOR metric", in Proc. of 1st International Workshop on Peer-to-Peer Systems (IPTPS), Mar. 2002.

[Minar01] Nelson Minar "Peer-to-Peer Harnessing the Power of Disruptive Technologies", ISBN 10: 0-596-00110-X | ISBN 13:9780596001100, 2001.

[Moon99] Sue Moon, Paul Skelley, and Don Towsley "Estimation and Removal of Clock Skew from Network Delay Measurement", in Proc. of IEEE INFOCOM '99.

[MZhang04] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. "PlanetSeer:

Internet Path Failure Monitoring and Characterization in Wide-Area Services", in Proc. of Sixth Symposium on Operating Systems Design and Implementation December 2004.

[NetflowWeb] http://www.cisco.com/en/US/products/ps6645/products_ios_protocol_option_home.html

[NapsterWeb] Napster - http://www.napster.com/

[Harvey03] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman, "SkipNet: A Scalable Overlay Network with Practical Locality Properties", in Proc. of the Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03), Seattle, WA. March 2003

[NISTWeb] http://snad.ncsl.nist.gov/itg/nistnet/

[NSWeb] http://www.isi.edu/nsnam/ns/

[NTPWeb] http://www.ntp.org/

[OpNetWeb] http://www.opnet.com/

[P2psimWeb] http://pdos.csail.mit.edu/p2psim/

[Padmanabhan05] Venkata N. Padmanabhan, Sriram Ramabhadran, Jitendra Padhye "NetProfiler: Profiling Wide-Area Networks Using Peer Cooperation", in Proc. of 4th International Workshop, IPTPS 2005

[Paisley06] J. Paisley and J. Sventek, "Real-time Detection of Grid Bulk Transfer Traffic", in Proc. of the 10th IEEE/IFIP Network Operations Management Symposium, Vancouver, Canada, April 2006

[Paxson96] Vern Paxson, "Towards a framework for defining Internet Performance metrics", in Proc. of INET'96, Montreal, 1996

[Paxson98] Vern Paxson "On Calibrating Measurements of Packet Transit Times", in Proc. of SIGMETRICS 1998

[PeerSimWeb] http://peersim.sourceforge.net/

[Pendarakis01] D. Pendarakis, S. Shi, D. Verma, and M.Waldvogel. "ALMI: an application level multicast infrastructure", in Proc. of 3rd Usenix Symposium on Internet Technologies and Systems (USITS 2001), March 2001.

[Pezaros04] Pezaros, D., P., Hutchison, D., Garcia, F., J., Gardner, R., D., Sventek, J., S., "In-line Service Measurements: An IPv6-based Framework for Traffic Evaluation and Network Operations", in Proc. of the 2004 IEEE/IFIP Network Operations and Management Symposium (NOMS'04), Seoul, Korea, April 19-23, 2004.

[Prieto06] A. Gonzalez Prieto and R. Stadler, "Adaptive Distributed Monitoring with

Accuracy Objectives", ACM SIGCOMM workshop on Internet Network Management (INM 06), Pisa, Italy, September 11, 2006.

[Preiss98] Preiss, Bruno (1998), “Data Structures and Algorithms with Object-Oriented Design Patterns in C++”, John Wiley & Sons, ISBN 0-471-24134-2

[Ratnasamy01] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, “Application-levelmulticast using content-addressable networks”, in Proc. of NGC, 2001.

[Rhagwan04] R.Rhagwan, P. Mahadevan, G. Varghese, and G.M. Voelker. “Cone: A Distribtued Heap-Based Approach to Resource Selection”, Technical Report CS2004-078, UCSD, 2004.

[RIPEWeb] http://www.ripe.net/

[RMIWeb] http://java.sun.com/developer/Quizzes/rmi/

[RMONWeb] http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/rmon.htm

[Rodrigues04] L. Rodrigues, J. Pereira,U. Minho,U. Lisboa “Self-Adapting Epidemic Broadcast Algorithms”, in Proc. of Workshop on Future Directions in Distributed Computing, June 2004, Bertinoro (Forlì), Italy.

[Rowstron01] Antony Rowstron, Peter Druschel “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.” in Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pages 329-350, November, 2001.

[Rowstron01+] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, “Scribe:The design of a large-scale event notification infrastructure” in Proc. of NGC, Nov. 2001.

[Schmidt03] Schmidt, C. Parashar, M. “Flexible Information Discovery in Decentralized Distributed Systems”, in Proc. of 12th IEEE International Symposium High Performance Distributed Computing, 2003.

[Saltzer84] J.H. Saltzer, D.P. Reed and D.D. Clark "End-to-End arguments in system design", ACM Transactions in Computer Systems 2,4, Novermber, 1984, pages 277-288

[Shaikh04] Aman Shaikh, Albert Greenberg,”OSPF Monitoring: Architecture, Design and Deployment Experience” in Proc. of USENIX Symposium on Networked Systems Design and Implementation (NSDI), March 2004.

[SNMP98] W. Stallings “SNMP, SNMPv2, SNMPv3, and RMON 1 and 2 3rd Edition”, Addison Wesley, 1998. ISBN 978-0201485349

[Stoica01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, in Proc. of ACM SIGCOMM 2001, San Deigo, CA, August 2001, pp. 149-160.

[Stolarz01] Damien Stolarz “Peer-to-Peer Streaming Media Delivery”, in Proc. of First International Conference on Peer-to-Peer Computing (P2P'01), Lingköping, Sweden, August 27 - 29, 2001

[Tabourier73] Tabourier, Y “All shortest distances in a graph: an improvement to Dantzig's inductive algorithm”, Discrete Math. 4, 83-87 (1973)

[TcpdumpWeb]http://www.tcpdump.org/

[Triantafillou03] P. Triantafillou N. Ntarmos S. Nikoletseas P. Spirakis “NanoPeer Networks and P2P Worlds”, in Proc. of Third International Conference on Peer-to-Peer Computing (P2P'03) Linköping, Sweden, September 01 - 03, 2003.

[Vakali03] Athena Vakali, George Pallis, "Content Delivery Networks: Status and Trends". IEEE Internet Computing 7(6): 68-74 (2003)

[VanRenesse03] Robert Van Renesse, Kenneth P. Birman “Astrolable: a robust and scalable technology for distributed system monitoring, management, and data mining”, in Proc. of TOCS, 2003

[VanRenesse04] Robbert van Renesse, Adrian Bozdog, “Willow: DHT, Aggregation, and Publish/Subscribe in One Protocol”, In Proc. ofIPTPS, 2004

[Waitzman88] Waitzman. D., Partridge, C. and Deering, S., "Distance vector multicast routing protocol", RFC 1075 1988

[Waxman98] B. M Waxman “Routing of multipoint connections”. IEEE JSAC, 6(9):1617–1622, 1988.

[White02] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. “An Integrated Experimental Environment for Distributed Systems and Networks” In Proc. of the 5th Symposium on Operating Systems Design and Implementation, Boston, MA, December 2002

[Winter87] P. Winter “Steiner Problem in Networks: A Survey”. Networks, 17(2):129–167, 1987.

[Yalagandula04] Praveen Yalagandula, Mike Dahlin “A scalable Distributed Information Management System”, in Proc. of SIGCOMM’04, Portland, Oregon, USA,

[Yao75] Yao, A.C.C.: “An O([Elloglog[V[) algorithm for finding minimal spanning tree”. Information Lett. 4, 21-23 (1975)

[ZebraWeb] http://www.zebra.org/

[Zelikovsky93] A. Zelikovsky “An 11/6-approximation Algorithm for the Network Steiner Problem”. Algorithmica, 9:463–470, 1993.

[Zhang02] Li Zhang, Zhen Liu and Cathy Honghui Xia, “Clock Synchronization Algorithms for Network Measurements," in Proc. of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2002) pp. 160-169

[Zhang03] Z. Zhang, S.-M. Shi, and J. Zhu, “SOMO: Self-Organized Metadata Overlay for resource management in p2p DHT,” in Proc. of the Second Int. Workshop on Peer-to-Peer Systems, (IPTPS’03), Berkeley, CA, Feb. 2003.

[Zhao04] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz “Tapestry: A Resilient Global-scale Overlay for Service Deployment” IEEE Journal on Selected Areas in Communications, January 2004, Vol. 22, No. 1, Pgs. 41-53.

[Zegura96] Ellen W. Zegura, Ken Calvert, S. Bhattacharjee “How to Model an Internetwork”, in Proc. of IEEE Infocom '96, San Francisco, CA.

[Zheng04] Pei Zheng, Lionel M.NI "EMPOWER: A Cluster Architecture Supporting Network Emulation", IEEE Transactions on Parallel and Distributed Systems, Volume 15, Issue 7, July 2004

[Zheng06] Changxi Zheng, Guobin Shen, Shipeng Li, and Scott Shenker, “Distributed Segment Tree: Support of Range Query and Cover Query over DHT,” in Proc. of the 5th International Workshop on Peer-to-Peer Systems (IPTPS-2006), Feb 27-28, 2006, Santa Barbara, USA.

[Zhuang01] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, “Bayeux:An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination”, in Proc. of NOSSDAV, June 2001.