NETWORK SUPPORT FOR MULTIMEDIA APPLICATIONS USING THE NETLETS ARCHITECTURE

By
Kalaiarul Dharmalingam

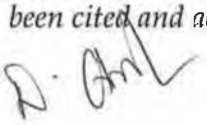
RESEARCH SUPERVISOR
DR. MARTIN COLLIER, BEng MEng PhD MIEEE

A THESIS PRESENTED FOR THE AWARD OF
DOCTOR OF PHILOSOPHY

February 2004

SCHOOL OF ELECTRONIC ENGINEERING
DUBLIN CITY UNIVERSITY

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy *is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.*



Signed:

ID number: 99141752

Date: 23 - FEBRUARY - 2004

Abstract

Multi-party multimedia networking applications such as e-commerce, distributed data analysis, Internet TV and advanced collaborative environments feature stringent end-to-end Quality of Service (QoS) requirement and require globally distributed user groups to be interconnected. The variety of delivery requirements posed by such applications are best satisfied using highly customised networking protocols. Hence, a demand for networks to migrate from the current *fixed service model* to a more flexible architecture that accommodates a wide variety of networking services is emerging.

New approaches are required in order to build such *service oriented* networks. Active networking is one such approach. Active networks treats the network as a programmable computation engine, which provides customised packet processing and forwarding operations for traffic flowing through network nodes. User applications can download new protocols into network elements at runtime, allowing rapid innovation of network services. This thesis makes the case for employing mobile agents to realise an active networking architecture, and describes such an architecture called the Netlets architecture. Netlets are autonomous, mobile components which persist and roam in the network independently, providing predefined network services.

This thesis presents the design and implementation of the Netlet node and the service deployment mechanisms that are required to distribute Netlet services in the network. Using the Netlet toolkit, variety of network services were designed to provide network support for multimedia applications in the Internet. A service was implemented to enhance the working of the RSVP protocol in order to provide robust end-to-end QoS support even when the network is only partially QoS provisioned. A scalable and reliable multicast protocol was implemented using the unicast communication model that accommodate heterogeneous receiver terminals. Another service integrates client-side server selection support into web sessions established over the Internet. A service was also developed which provides QoS signalling support to legacy applications.

It is shown that these Netlet services are of practical value using performance measurements to assess Netlet responsiveness. Netlet based solutions may be deployed using existing technologies to provide support for a wide range of multimedia applications in the Internet. The Netlets architecture has thus been shown to allow *value-added services* to be added to existing networks. By optimising the Netlet architecture implementation, this may be extended to services operating on high-speed (1Gb/s and upwards) links. It thus shows promise as an architecture for building the next generation of active networking solutions.

~ Dedicated to Mama, Nallaianandhan~
Appa, Amma and Ammama

Acknowledgement

While the cover page just includes my name, this work would not have been possible without the support of many people with whom I interacted during my time at DCU. First, I would like to thank my supervisor, Dr. Martin Collier, for all his support during this period of work. He taught me the basic quality required to get a PhD, i.e. to think autonomously. Furthermore, I would like to convey a special thanks for the effort Martin put in to make the thesis look a professional document.

I would like to thank the members of the examination panel, Dr. Ahmed Karmouch, Dr. Barry McMullin and Prof. David Cameron. I would like to thank all the members of the switching and systems laboratory for all their support. I would like to specially thank my colleague Karol Kowalik, for discussions and collaboration.

I thank my uncle, Nallaianandhan Chelliah, for motivating and believing that it was always possible for me to achieve this. I would like to thank my parents and grandparents for their support and understanding. I would like to convey a special thanks to Srigengan Balasingham for his guidance and support. I would like to thank Prince Anandarajah, for proof-reading some key chapters of my thesis. Finally, I would like to thank all my friends and family members for their support during this period of work. Thank you one and all.

List of Publications

- **Kalaiarul D** and Martin Collier. Netlets: A New Active Network Architecture. *In Proceedings of IEE/IEI Symposium on Telecommunications Systems Research,* Dublin, Ireland, November 2001.
- **Kalaiarul D** and Martin Collier. An Active Network Solution to the Problem of RSVP Reservation Gaps. *In Proceedings of IEE London Communication Sympoisum (LCS),,* London, UK, September 2002.
- **Kalaiarul D** and Martin Collier. Transparent QoS Support For Network Applications using Netlets. *In Proceedings of IEEE/IFIP Mobile Agents for Telecommunication Applications (MATA),* pages 206–215, Barcelona, Spain, October 2002.
- **Kalaiarul D** and Martin Collier. Transparent and Scalable Client-side Server Selection using Netlets. *In Proceedings of IEEE International Conference on Open Architectures and Network Programming, (OPENARCH),* pages 120–129, San Francisco, CA, USA, March 2003.
- **Kalaiarul D**, Karol Kowalik and Martin Collier. RSVP Reservation Gaps: Problems and Solutions. *In Proceedings of IEEE International Conference on Communications (ICC),* pages 1590–1595, Anchorage, Alaska, USA, May 2003.
- **Kalaiarul D**, Karol Kowalik and Martin Collier. Reservation Gaps. *Submitted to Journal of Elsevier Computer Communications.*
- **Kalaiarul D** and Martin Collier. MENU: Multicast Emulation using Netlets and Unicast. *Submitted to IEEE GLOBECOM* 2004.

Contents

List of Figures ii

List of Tables v

1 Introduction 1
- 1.1 Network Programmability . . . 2
- 1.2 Netlets Network . . . 3
- 1.3 Thesis Objectives . . . 3
- 1.4 Contributions of this Thesis . . . 4
  - 1.4.1 Netlets Network Architecture . . . 4
  - 1.4.2 Multimedia Applications of Netlets . . . 5
- 1.5 Outstanding Issues . . . 7
- 1.6 Organisation of the Thesis . . . 7

2 State of the Art 9
- 2.1 Barriers to Network Evolution . . . 9
- 2.2 Programmable Networks . . . 11
  - 2.2.1 Programmable Node . . . 12
  - 2.2.2 Active Networks . . . 14
  - 2.2.3 Programmable Node vs. Active Networking . . . 15
- 2.3 Packet Processing Support in the Current Internet . . . 16
- 2.4 Applications of Active Networks . . . 17
- 2.5 Active Networks: Themes and Concepts . . . 25
- 2.6 Active Networking Models . . . 27
  - 2.6.1 Active Packet Models . . . 28
  - 2.6.2 Active Node Models . . . 30
  - 2.6.3 Active Packets vs. Active Node Models . . . 35
- 2.7 Mobile Agents . . . 37

2.7.1 Mobile Agent Systems . . . 39
2.7.2 Mobile Agents for Networking Applications . . . 42
2.7.3 Mobile Agents and Active Networking . . . 43
2.7.4 Why use Mobile Agents for Active Networking? . . . 44
2.7.5 Why Netlets? . . . 46
2.8 Summary . . . 49

3 Netlets Network **50**
3.1 The Netlets Network Architecture . . . 50
3.1.1 Architecture of a Netlet Node . . . 51
3.1.2 Netlet Distribution . . . 53
3.2 Implementation of the Netlet Runtime Environment . . . 54
3.2.1 Netlet Deployment . . . 58
3.2.2 Packet Processing in the Netlets Network . . . 65
3.3 Netlet Deployment . . . 68
3.3.1 Proactive Service Deployment . . . 68
3.3.2 Reactive Service Deployment . . . 72
3.4 Summary . . . 78

4 Netlets for Multimedia Applications **80**
4.1 RSVP Reservation Gaps . . . 81
4.1.1 The Problem . . . 81
4.1.2 RSVP Reservation Gaps . . . 82
4.1.3 The Role of Netlet Nodes . . . 84
4.1.4 Application Level Support . . . 84
4.1.5 Robust Reservation Support . . . 85
4.1.6 Routing Algorithms To Minimise The Number of Gaps Along A Q-Flow's Path . . . 88
4.1.7 Remarks . . . 89
4.2 Transparent and Scalable Client-side Server Selection using Netlets 90
4.2.1 The Problem . . . 90
4.2.2 Goals . . . 91
4.2.3 Solution Overview . . . 92
4.2.4 Dynamic Setup Of Virtual Primary Server . . . 92
4.2.5 Server Selection using Director Services . . . 95
4.2.6 Supporting Architectural Features . . . 96
4.2.7 Benefits of Employing Director Service Netlets . . . 100
4.2.8 Remarks . . . 101

4.3 MENU: Multicast Emulation using Netlets and Unicast 101
4.3.1 The Problem 101
4.3.2 Goals of the MENU Protocol 103
4.3.3 MENU Protocol Concept 104
4.3.4 Hot Spot Delegates 105
4.3.5 Hot Spot Delegates as Virtual Sources 106
4.3.6 MENU Protocol Details 107
4.3.7 Traffic Distribution from Server to HSDs 109
4.3.8 A Reactive Approach for Delivery Tree Construction 109
4.3.9 Dynamics of the MENU Protocol 112
4.3.10 Remarks 113
4.4 Transparent QoS signalling support to Network Applications . . . 114
4.4.1 The Problem 114
4.4.2 QoS Support using Netlets 115
4.4.3 Benefits of Using Netlet Services 119
4.4.4 Remarks 120
4.5 Summary 120

5 Evaluation of Proposed Applications **122**
5.1 Robust Reservation Support using Netlets 122
5.1.1 Unmanaged Vs Netlet Managed Gap 123
5.1.2 Routing Enhancements 126
5.1.3 Simulation Results 128
5.1.4 Remarks 131
5.2 Server Selection using Netlets 132
5.2.1 Client Perceived Service Response Time 133
5.2.2 Server Load Distribution 133
5.2.3 Experiments 134
5.2.4 Server Selection in the Internet 136
5.2.5 Remarks 140
5.3 Large Scale Deployment of MENU 140
5.3.1 Experiments 141
5.3.2 Network Model 141
5.3.3 Results 143
5.3.4 Remarks 148
5.4 QoS Support for Network Applications using Netlets 149
5.4.1 Experiment 149
5.4.2 Implementation 149

5.4.3 Deployment in Large Networks . . . 152
5.4.4 Remarks . . . 152
5.5 General Performance Characteristics of a Netlet Node . . . 153
5.5.1 Performance Evaluation for U-Netlet . . . 153
5.5.2 Performance Evaluation for M-Netlet . . . 160
5.5.3 Unicast-Netlet and Multicast-Netlet Services . . . 162
5.6 Dynamic Deployment of Netlet Services at Runtime . . . 163
5.7 Service Deployment Latencies . . . 165
5.7.1 Reactive Service Deployment . . . 165
5.7.2 Proactive Service Deployment . . . 169
5.8 Issues in Netlet Deployment . . . 172
5.9 Summary . . . 173

6 Conclusions **176**
6.1 Contributions . . . 176
6.1.1 Netlets Network . . . 176
6.1.2 Network Support for Multimedia Applications using Netlets 177
6.2 Future Work . . . 179
6.3 Concluding Remarks . . . 182

A An Analytical Model to Study the Reactive Service Deployment **183**
A.1 Analytical Model . . . 183
A.1.1 Reactive Deployment . . . 183
A.1.2 State Transitions at a Netlet Node . . . 183
A.1.3 M/G/1 Model with Setup Time . . . 184
A.2 Summary . . . 188

Index **189**

Bibliography **192**

List of Figures

2.1 From Monolithic Computers to The PC Paradigm 12
2.2 Programmable Node Vs. Active Networks 12
2.3 The IEEE P1520 Mapping for IP Routers 14
2.4 Architecture of an active node 26
2.5 Active Networking and the Mobile Agent Paradigm 44

3.1 A Simple Netlet Network Model 51
3.2 Architecture of a Netlet Node 51
3.3 Netlet Naming Scheme 54
3.4 Architecture of a Netlet Runtime Environment 55
3.5 API for building Netlets 56
3.6 API of the Netlet Management Engine 57
3.7 API for Packet Capture 58
3.8 Pseudo-code for Migration 59
3.9 Pseudo-code for Sending a Netlet 60
3.10 Pseudo-code for Receiving a Netlet 62
3.11 Pseudo-code for Autonomous Operation of a Netlet 63
3.12 Pseudo-code for Unpacking a Netlet 63
3.13 Pseudo-code for Starting a Netlet 64
3.14 Packet Processing at the NRE 65
3.15 Pseudo-code for Proactive Packet Processing 67
3.16 Pseudo-code for Reactive Packet Processing 69
3.17 Proactive Service Deployment 70
3.18 Algorithm for Active Node Discovery and Service Deployment . . 71
3.19 Packet Processing at a Netlet Node 73
3.20 Domain Level $L1$ Cache 74
3.21 An Example of Service Discovery in ABC Framework 76

3.22 Format of a Request Packet in Stigmergy 76

4.1 IntServ Over DiffServ QoS Model With Non-QoS Islands 82
4.2 The Reservation Gap in an IntServ Network 85
4.3 Replicated Servers and Communities of Interest 93
4.4 Address Sharing in the Netlet Scheme 94
4.5 Transparent Server Selection using Director Services 95
4.6 Cumulative Request Distribution Across Communities of Interest . 98
4.7 Algorithm for Active Node Discovery and Service Deployment . . 99
4.8 Two Level Hierarchical Model 105
4.9 Traffic Distribution From HSD to Users 110
4.10 End-to-End QoS Model: IntServ and DiffServ 115
4.11 QoS Support Using Netlets 116
4.12 QoS Support Request Interface 117

5.1 Experimental Setup for Robust Reservation Support 123
5.2 Service Degradation For QoS Flows across the Gap 124
5.3 Netlet Managed Reservation Gap 125
5.4 The ISP topology 126
5.5 Call blocking probability of SP, MR-S and S-MR under increasing number of QoS aware nodes 129
5.6 Average length of the path chosen by SP, MR-S and S-MR 130
5.7 Reliability of routing decisions of SP, MR-S and S-MR 131
5.8 Cost of path monitoring 132
5.9 Experimental Setup for Load Based Server Selection 134
5.10 Load Based Server Selection 136
5.11 Impact of Server Load on Service Response 137
5.12 Experimental Setup for Server Selection in the Internet 138
5.13 Comparison of Server Selection Metrics 139
5.14 Network Topology used for Multicast Gain Measurement 142
5.15 Multicast Gain with Active Stub Networks 144
5.16 Network Topology used for Stress Measurement 146
5.17 Link Stress 147
5.18 Reduction in Forwarding State 148
5.19 Error Recovery Delay for a Multicast Session in MENU 149
5.20 Experimental Setup 151
5.21 Experimental Setup for Measuring Performance in Unicast Communication 154
5.22 Forwarding Latency Across a U-Netlet Service 156

5.23 Processing Overhead in C-forward and U-Netlet Service Relative to Kernel Level Forwarding 157
5.24 Throughput of a U-Netlet Service 158
5.25 Throughput of the Netlet Forwarding Service and C-forward 159
5.26 Experimental Setup for M-Netlet Service 160
5.27 Forwarding Latency Across a M-Netlet Service 161
5.28 Throughput of a M-Netlet Service 162
5.29 Experiment Setup To Evaluate the Reactive Service Deployment Scheme 164
5.30 Network Topology used for Analysing Stigmergy 166
5.31 Service Deployment Delay in the Reactive Model 167
5.32 Service Deployment Time for various Cache Locations within the Network 168
5.33 Normalised Delay of the Cache-less Approach for Service Deployment 169
5.34 Network Topology used for Service Deployment 170

A.1 State Transitions at a Netlet Node 184

List of Tables

5.1 Forwarding latency for a packet of 512 Bytes in length when supporting dynamic deployment 165
5.2 Delay to Discover Active Nodes at various Geographical Locations 171

A.1 Netlet Node states and the M/G/1 queuing model with vacation and setup times 185

Chapter 1

Introduction

The unparalleled success of the current Internet, especially since the advent of the World Wide Web, makes it the technology of choice for building a unified global communication infrastructure. Considering the next generation of Internet technologies, a paradigm shift from the traditional point-to-point communication model is emerging, which involves multi-party and multimedia connections. Some examples of such media rich multi-party applications are audio and video broadcasting, distributed data analysis, virtual reality games, and collaborative environments.

Yet, despite the great promise of various emerging technologies, progress towards wide scale deployment is slow, in part because the infrastructure is inflexible. The networks of today were designed for a fixed service model, whereby network elements (e.g. switches, routers) are closed boxes that permanently house and execute a restricted set of vendor software.

The rate of change in today's networks is restricted by slow standardisation processes and compatibility concerns. The result is that the introduction of new services occurs much slowly than the emergence of new applications and technologies that benefit from them. The present backlog of networking services such as IPv6 [1], RSVP [2], IP Multicast [3] testifies to this fact.

Another feature which hampers the current Internet from providing support

for high end networking applications is its best-effort nature. In the current Internet there are no guarantees on delivery and timeliness of data transfer. However, media rich applications require strict guarantees on the end-to-end service levels provided by networks. Finally, due to the continuing expansion in the size of the Internet, it has become increasingly difficult to manage and maintain the network. Therefore, the deployment time for new protocols are also increasing.

In order to accommodate various environments, applications and traffic workloads, networks should support a wide variety of protocols and work with various service level requirements so as to cater for individual application demands. Consequently, the network should play an active role in supporting the needs of the applications and end user demands. Additionally, there is also a need for more autonomous computers and network elements that are able to follow the accelerated pace of evolution in applications and release users from tedious and error prone software management tasks.

1.1 Network Programmability

The need to rapidly develop and deploy new services has instigated the need to revolutionise the way networking systems are built. One way to overcome the network evolution problem is to introduce programmability into network nodes, a feature already available in end user systems.

Two proposed approaches to support network programmability are: (i) Programmable Nodes [4]; and (ii) Active Networks [5]. The Programmable Node approach uses a set of open programmable network interfaces to provide controlled open access to switches, routers and base stations.

The active networks [5] approach involves the placement of user-defined computation at network elements, thereby enabling customised processing of data inside the network. Existing active network models use mobile code to support the dynamic deployment of new services at runtime into network nodes.

Alternatively, mobile agents [6] can be used to build active networking systems. A mobile agent based active network architecture provides a unified framework for service deployment and network management purposes.

1.2 Netlets Network

The Netlets network architecture [7] is an active network infrastructure using the mobile agent paradigm. Netlets are autonomous, nomadic mobile components which persist and roam in the network independently, providing predefined network services. The Netlets concept involves demand based distribution of network services. Popular network services are widely replicated across the network automatically, while instances of obsolete or unsuccessful services are removed gradually.

The initial work on Netlets [7] presented the general concepts and the potential benefits of such an architecture. This thesis addresses the use of Netlets to provide network support for multimedia applications in the Internet.

1.3 Thesis Objectives

The main objectives of this thesis are:

- **Architectural Level** - *to implement the Netlets architecture in such a way that network programmability is introduced to augment and support existing networking protocols, thereby providing new networking services.*
- **Network Level** - *to define mechanisms which support Netlet deployment in wide area network environments, such as the Internet.*
- **Application Level** - *to demonstrate the suitability of such an architecture to provide support for multimedia communications in the Internet.*

Methodology: I have built a prototype of the Netlet node to evaluate the concepts that are put forward in this thesis. This prototype includes the implementation of the Netlet execution environment and the Application Programming Interface (API) required to build Netlet services. I use the Java language for this purpose. I have evaluated the performance of the prototype and have demonstrated its applicability for a variety of multimedia applications. For evaluation of the applications, I use testbeds constructed in a laboratory environment using networked PCs. Due to the difficulty of implementing a large network and generating various traffic patterns in the laboratory, simulations are used wherever appropriate to evaluate the developed services.

1.4 Contributions of this Thesis

The contributions of this thesis include: (i) the design of the Netlet node and the service deployment mechanisms that are required to distribute Netlets in the network; and (ii) the design of a range of new networking services using Netlets, which are used to provide network support for a variety of multimedia applications in the Internet.

1.4.1 Netlets Network Architecture

Netlet Node Architecture: As a first step, I present the design of the Netlet node. Following this, I present the implementation of the Netlet Runtime Environment (NRE). The NRE provides the interface to dynamically "plug-in" Netlet services to a network node.

Stigmergy Protocol: The Netlets architecture follows a decentralised approach for service distribution. Thus, the nodes at which services are located are not known *a priori*. I propose a service discovery protocol, referred to as Stigmergy, which supports the discovery of active services distributed across the Internet.

A Scheme to Discover Active Nodes in the Internet: Due to the heterogeneous nature of the Internet not all nodes in the Internet will support Netlet services. Hence, mechanisms to discover active nodes are required. I propose a DNS-based discovery scheme which allows Netlet services to locate active node support available in the network.

1.4.2 Multimedia Applications of Netlets

I have designed a variety of network services using Netlets, which provide network support for multimedia applications in the Internet. One service enhances the working of the RSVP protocol in order to provide robust end-to-end QoS support even when the network is only partially QoS provisioned. Another service provides a scalable and reliable multicast using the unicast communication model that accommodates heterogeneous receiver terminals. I have also implemented a service which transparently integrates client-side server selection support into web sessions established over the Internet. Finally, a service has been developed to provide QoS signalling support mechanisms to end applications in a transparent manner. The following discussion expands on these contributions.

Solutions to the Problem of Reservation Gaps: High-end networking applications such as e-commerce, multimedia, distributed data analysis and advanced collaborative environments feature demanding end-to-end quality of service (QoS) requirements. Due to the heterogeneity exhibited by the Internet, a route from source to destination for such a flow may not be available which is comprised exclusively of QoS supporting path segments. Hence the flow must traverse one or more non-QoS path segments referred to here as *reservation gaps*. I have studied the problem of reservation gaps and presented solution using Netlets. Furthermore, to improve the reliability in path selection and to minimise the influence of reservation gaps along the path of a QoS flow, I propose two new routing al-

gorithms[1], *the most reliable – shortest path (MR-S)* algorithm and *the shortest – most reliable path (S-MR)* algorithm, that select paths with the minimum number of reservation gaps.

Transparent QoS support of Network Applications using Netlets: End-to-end QoS support is deemed necessary to support multimedia communication in the Internet. Hence, mechanisms to enable end applications to request desired QoS levels from underlying networks is important. In contrast, there exists a large pool of non-QoS aware applications that are unable to exploit and benefit from the QoS support available in networks. Furthermore, the technology to support QoS in networks is not yet fully mature. Thus, developing an application to interact with a specific QoS protocol carries the danger that the application may become obsolete if the QoS protocol is modified or superseded. I propose a novel approach based on Netlets to transparently retrofit QoS support to legacy network applications.

Transparent Client-Server Selection using Netlets: Content replication in the Internet has been found to improve the service response time, performance and reliability of web services. When working with such distributed server systems, the location of servers with respect to client nodes is found to affect the service response time perceived by clients in addition to server load conditions. This is due to the characteristics of the network path segments through which client requests get routed. Most server selection methods [9–14] proposed to date work to distribute load across servers. I propose a novel technique to support transparent and flexible client-side server selection in the Internet using Netlets.

Multicast Emulation using Netlets and Unicast (MENU): Large scale multiparty applications such as Internet TV and software distribution have generated a demand for multicast services to be an integral part of the network. This will allow such applications to support data dissemination to large groups of users

[1]developed jointly with my colleague Karol Kowalik [8]

in a scalable and reliable manner. Existing IP multicast protocols [3, 15–18] lack these features and also require state storage in the core of the network which can be costly to implement. I propose a new multicast protocol referred to as MENU, which overcomes these shortcomings.

1.5 Outstanding Issues

Two key areas in active network research are critical to the successful deployment of Netlets. These concern resource management and security.

In traditional networks, bandwidth is usually the primary network resource that is shared among traffic flows at a network node. However, in active networking both link and node level resources will have to be managed. Node resources include, for example, memory and CPU cycles. In order to ensure fair access and to avoid the abuse of networking resources, active network nodes will have to impose strict limits on resource usage by third-party services.

Another major impedient to the wide spread deployment of active networks is the concern over safety and security. Since users can dynamically load network services at runtime into network elements, there is potential for hostile users to launch malicious network services. Hence, robust safety and security mechanisms will have to be present in network nodes.

These issues lie outside the scope of this thesis, although it is recognised that solutions to these problems must be found if the Netlet architecture is to be deployed in "live" networks.

1.6 Organisation of the Thesis

Chapter 2 presents a survey of the current research efforts in the field of active networks. Next, I discuss a wide range of applications that demonstrate the potential benefits of active networking. A survey of the different active network

approaches and available prototype models is then presented. Finally, I present the shortcomings of existing active networking models and my reasons for concluding that mobile agents are a suitable paradigm to build active networking systems.

Chapter 3 presents the Netlets architecture and the deployment mechanisms employed to distribute Netlet services in the network. I present the API and the associated library of methods as well as implementation mechanisms to support the execution of Netlet services at network nodes. The remainder of this chapter discusses reactive and proactive service deployment schemes for the distribution of Netlet services in the network.

Chapter 4 presents Netlet based solutions to a variety of problems that occur when supporting multimedia applications in the Internet. The areas taken up for investigation includes: (i) Quality of Service (QoS); (ii) Multicast; and (iii) Server Selection. These solutions are intended to ensure a graceful migration from best-effort model to an Internet technology with multimedia support.

Chapter 5 presents results from experiments that were carried out to evaluate the set of applications described in Chapter 4. To evaluate the practicality of employing the Netlet prototype for a wider set of applications, I conducted tests to analyse the performance and service deployment characteristics of the Netlets architecture, the results of which are also presented here. Finally, I present a set of general conclusions drawn from my experience on the work on Netlets.

Chapter 6 summarise our contributions and describes some suggestions for future work in developing the Netlets architecture.

Chapter 2

State of the Art

This chapter sets the stage for the research carried out in this thesis. Presented first are the limitations of the current networking model and the need to migrate to a new networking paradigm. Next active networks are described and their suitability for building a flexible networking architecture is assessed. Following this, I discuss a wide range of applications that demonstrate the potential benefits of active networking. Next, a survey of the different active network approaches and available prototype models are presented. Finally, I present the shortcomings of existing active networking models and the benefits of employing mobile agents to realise active networking systems.

2.1 Barriers to Network Evolution

The current Internet suffers from slow network evolution. Furthermore, it requires a lot of human interaction for network operation, maintenance and management. The major factors that contribute to these problems are:

Monolithic Network Elements: The network elements of today (e.g. routers, switches) are closed vertically-integrated systems in which the hardware and software segments are tightly coupled to each other. Due to this feature, network elements only support configuration and management features largely limited to

the set of pre-installed options available at the time the element is shipped. Extending these requires upgrades and bug fixes, which are costly and difficult to implement. This situation is similar to the case of strictly configurable computers that were available in the early 1970s. For example, the non-adaptable nature of the current network model has clearly delayed the deployment of multicast routing protocols [3], even though practical solutions exist for implementing multicast support in the Internet.

Increased Redundancy and Added Complexity Levels in Network Protocols: In the current networking model, a new protocol can often only be supported by installing new networking equipment that supports it. This is a costly process. Due to this problem, not all elements in a network might ubiquitously support a new/modified version of a protocol (e.g. a revised version of a multicast protocol such as DVMRP [15]). Hence to support interoperability with legacy protocol stacks backward compatibility between releases or updates is required; this increases the overall complexity level of protocols. Furthermore, during migratory periods obsolete versions of protocols will coexist with new/modified versions of the protocols, thereby imposing redundancy at network nodes.

General-Purpose Protocols: The business models and priorities of equipment vendors and service providers vary significantly. Vendors want to cater for a large customer base with the available resources (e.g. technical expertise), while individual service providers want network equipment with a customised set of services which suits their requirements. Furthermore, the range of services required by individual service providers are large and changes over time. To overcome this problem, vendors bundle a set of general purpose protocols that satisfy the common needs of any service provider. Under this model, the service provider may not receive the functionality and versatility required to satisfy their business models. Employing general-purpose protocols for a wide variety of applications has been found to affect the performance of network applications [19].

Deadlock in Protocol Evolution: In today's business environment, vendors delay commercial implementation of any new protocol pending a clear demand for the service. In contrast, without actual availability of the protocol, there will not be enough users to create a critical mass that will increase the demand for the service. This causes a deadlock situation, thus inhibiting protocol evolution. IPv6 [20] and IP Multicast [3] are some of the critical protocols that have suffered from this problem.

Protocol Standardisation: The current network model achieves interoperability through protocol standardisation. It is not unusual that by the time a standard is produced, technological development and user demand may have rendered it largely obsolete. In summary, changing network protocols in the current network model is lengthy and difficult.

Manual Network Management: Network management in the current network model is performed manually. Network managers employ special scripting languages and large sets of configuration variables, typically the Management Information Base (MIB), for this purpose. With the exponential growth in the number of network nodes, network management and configuration has become costly and time consuming. It has been estimated that one-third to one-half of a company's total IT budget is spent on network management [21]. Furthermore in the current model, network elements have to be brought off-line for service upgrades or bug fixes. Such service disruptions are costly and inefficient.

2.2 Programmable Networks

The need to rapidly develop and deploy new services can only be addressed if we revolutionise the way networking systems are built. It has been identified that by replacing the closed vertically-integrated systems of today with programmable network nodes, it will be possible to realise a flexible networking architecture. The phenomenal success of the PC paradigm testifies to this fact

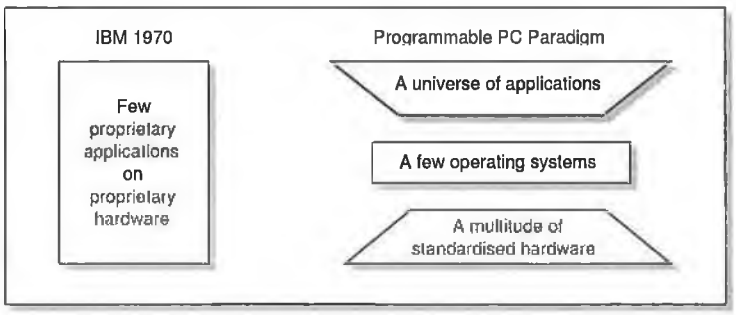


Figure 2.1: From Monolithic Computers to The PC Paradigm

(Fig. 2.1 [22]). Two major schools of thoughts have been proposed for the realisation of the programmable networking paradigm. They are (i) the Programmable Node approach [4]; and (ii) the active networking approach [5]. Fig. 2.2 shows the different approaches taken by these models.

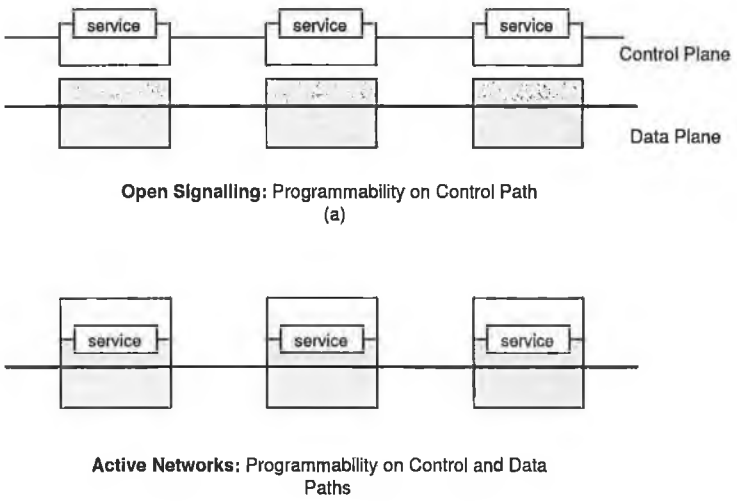


Figure 2.2: Programmable Node Vs. Active Networks

2.2.1 Programmable Node

The idea behind the Programmable Node approach is that by modelling the communication hardware using a set of open programmable network interfaces, open access to switches, routers and base stations can be provided. These open interfaces allow service providers and network operators to manipulate the states

of the network nodes through the use of middleware toolkits so as to construct and manage new services. By building an API-based network model, service providers, independent software vendors and other developers in the IT and telecommunication industries will be able to participate in rapid network evolution. This separation between the network hardware and the control algorithms is referred to as virtualisation [23].

The xbind broadband kernel [24] is an example of a Programmable Node architecture. The xbind service architecture is based on a distributed component based programming paradigm that allows modular construction of multimedia services. It includes components to implement mechanisms for broadband signalling, switch control and management, and distributed resource allocation.

IEEE P1520 [25] was initiated by the OpenSig community [4] as an effort to define and standardise software abstractions of network resources and provide application programming interfaces (APIs) for the manipulation of these resources. The P1520 interfaces are strictly layered and are named the V-(value added), U-(user), L-(lower) and CCM-(connection control and management) interfaces (Fig. 2.3-a). These open interfaces allow service providers and network operators to manipulate the states of the network through the use of middleware toolkits in order to construct and manage new network services.

Another effort in standardising programmable node interfaces is being pursued by the Multiservice Switching Forum[1]. The major goal of this group is to define an architecture separating control and data planes that facilitates the introduction of new network services over ATM-capable networks. The SoftSwitch Consortium[2] shares similar goals in IP network environments. Similar efforts are also being pursued by the Parlay[3] and JAIN[4] groups.

[1]http://www.msforum.org/
[2]The SoftSwitch consortium http://www.softswitch.org/.
[3]PARLAY http://parlay.msftlabs.com
[4]JAIN http://java.sun.com/products/jain/

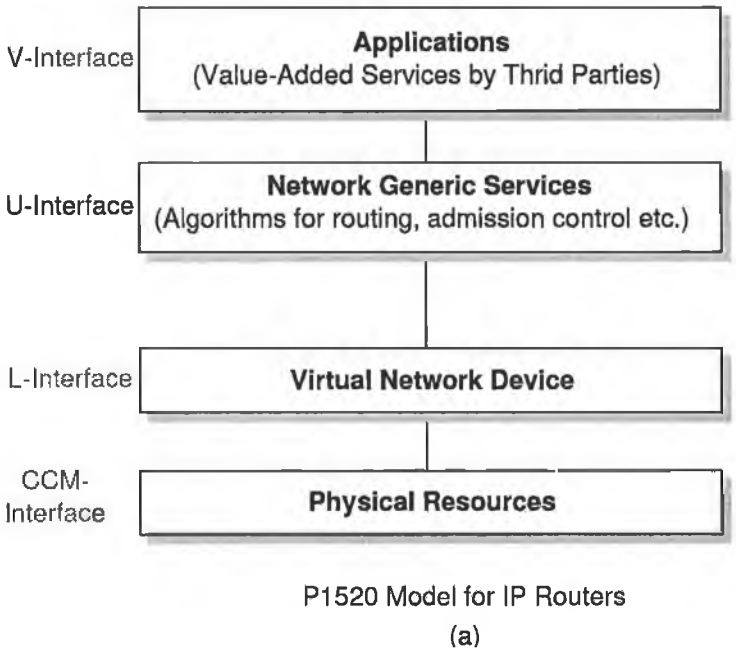


Figure 2.3: The IEEE P1520 Mapping for IP Routers

2.2.2 Active Networks

Active networking views the network as a programmable computation engine, which provides customised packet processing and forwarding operations for traffic flowing through them (see Fig. 2.2). In active networking, the traditional model of packet forwarding, *store-and-forward*, is replaced by *store-compute-forward*, thereby enabling packet processing support at intermediate nodes as they travel through the network.

A key feature of this approach is the flexibility to dynamically deploy new services at network nodes in response to user demands. This approach is motivated by both the trend towards network services that perform user-driven computation at intermediate nodes and the emergence of mobile code technologies that make network programmability possible.

2.2.3 Programmable Node vs. Active Networking

Both Programmable Node and active network approaches, overcome the problem of the deadlock situations in protocol evolution that are present in the current Internet. Furthermore these models provides a scalable and cost-effective means to test new networking services without requiring co-operation from multiple vendors, thereby increasing the scope for network evolution.

Considering the Programmable Node model, the programmability is restricted to the control plane and targeted towards connection-oriented networks only. This model supports switch control only on the connection level and not the packet level. In comparison, active networks are intended to supporting programmability in datagram networks. Furthermore, they support packet processing both on the data path and control path, thus building a more flexible networking architecture.

In traditional networks, routers permanently house a predefined set of protocols regardless of the level of demand for these. This increases the cost of network nodes due to the requirement to support redundant services. Due to the feature of dynamic service provisioning available with active networks, permanent storage of protocols that are seldom used would not be required at every node on the network.

In the Programmable Node approach, the set of services provided at a node are limited, and thus so are its capabilities. In contrast, active networks enable rapid deployment of new network protocols in response to user demand. This is due to the fact that only those network nodes that are required to host the service need to agree on the service definition and install it.

Finally, the Programmable Node model relies on human intervention for service installation and management, which can be costly and time consuming, considering the ever increasing size of the networks. However, with its ability to integrate services at runtime, active networking reduces the complexity of net-

work management tasks. Due to the benefits exhibited by the active networking approach over that of the Programmable Node model, we take up the former for further investigation.

2.3 Packet Processing Support in the Current Internet

While the concept of active networks may seem a radical departure from the traditional networking paradigm which is based on the end-to-end argument, it is in fact a natural evolutionary change. Considering the reality of the current network environment, it is evident that current networks provide enhanced processing beyond packet forwarding. Some well known examples of such services that do not typically belong in the set of traditional router level functions include firewalls, load balancers, media gateways, Network Address Translators (NATs), Application Level Gateways (ALGs), packet tunnelling and Differentiated Services (Diffserv).

Anther recent trend is that vendors have started building network equipment that accommodates operation above Layer 3. An example of such equipment is Softswitch from Lucent Technologies[5] which supports converged data, voice and multimedia services platform for wireline and wireless networks. In addition, application specific network level services such as voice codecs for VOIP support in Cisco routers and email virus-scanners [6] have started appearing as permanently integrated services into commercial network elements.

The potential and benefits that result from supporting network based processing has been clearly identified. However, due to the limitations that exist with the current networking model (see section 2.1), there is an increasing trend to build ad-hoc solutions so as to cater for individual application demands. The emergence of overlay networks [26] is an example that testifies to this fact. The

[5]Lucent Technologies APX 8000
[6]http://www.blazenet.com/

problem with the such a network model is that it replicates lower layer functionality, often with degraded performance levels. In contrast, active networks define a general and extensible network model on which a wide variety of network services can be built.

2.4 Applications of Active Networks

The successful deployment of active networks relies on the development of applications and services that demonstrates its potential benefits. Below, we present a wide range of applications that benefit from active network support. This discussion includes applications from such fields as: (i) network management, (ii) security, (iii) caching, (iv) data transcoding, (v) congestion control and (vi)multicasting.

Network Management

Network Management in the current network environment is achieved by polling the managed devices from the management stations for data, looking for anomalies. However, with the increase in the number of network nodes, this traditional technique becomes problematic. The primary reasons is that this technique concentrates intelligence of the whole network management system within a few management stations at the network edges. These network management stations become points of implosion in the whole network. Furthermore, the poll-and-check approach severely limits the ability of the network to track problems in a timely and efficient manner. These effects are mitigated in a hierarchical implementation but scalability would be enhanced using a distributed implementation.

Goldszmidt *et al.* [27] deployed active network services at the managed nodes to perform distributed network management tasks. This approach, by delegating management processes to the target node itself, overcomes implosion problems at management stations. The amount of redundant information communicated

to the management stations is reduced thereby increasing the goodput between the managed elements and the management station.

The Regatta framework [28] realises an automated management system for supervision of network clouds by using the active node support present in the network. In this model, active services which embed fault diagnosis processes and contingency plans are deployed at those network nodes which require management. On identifying fault or anomalies, the active services execute the in-built contingency plans and then report the results back to the management stations. This approach to network management avoids the need for continuous human intervention for network operation and maintenance.

Another problem with traditional network management systems is the lack of support for scalable accounting infrastructures. In the current model, network nodes collect data regarding the traffic flowing through them and upload them to centralised accounting servers for processing. Following this, these servers process the accumulated data and extract the required information, which are later used by accounting applications for billing purposes. The problem with such an approach is that: (i) the processing time required to extract the accounting information from the raw data set is huge (in terms of hours/days) [29]; and (ii) the continuous transfer of raw data from network nodes to accounting servers consumes a large amount of network resources.

Travostino *et al.* [29] proposed to co-locate processing engines with network nodes so as to support customised operations on accounting data exactly where they are being generated. In this model, new accounting tasks are integrated into the processing engines using active services referred to as "plugins". This architecture improves the responsiveness, scalability, and dependability of accounting applications (e.g. real time billing and traffic monitoring) to the benefit of service providers.

The idea of providing virtual private networks (VPN) using public network

infrastructures such as the Internet has been identified to be cost-effective [30]. However, due to the closed nature of the current network model, provisioning such support is cumbersome. Rebecca *et al.* [31] proposed a scheme to build VPNs on-demand by dividing the resources of individual network nodes into one or more logically separate active network services, referred to as "switchlets". In this model, a full-blown virtual network is constructed dynamically by acquiring switchlets in one or more of the network nodes along the end-to-end path.

Network Security

Existing network intrusion detection approaches are passive; i.e. they are only able to statically define mechanisms for defending against such attacks. For example, with the case of Distributed Denial of Service (DDoS) attacks, one of the primary targets are network routers.

Existing routers do not include support for automatically detecting and defending against such attacks. In the existing network model, when network administrators identify attack patterns, they manually add packet filters or rate limiters to the routers under attack, in order to prevent further damage. With the increase in both the number of network nodes and the wide variety of attacks (e.g. Code Red[7]), relying on human intervention can be costly.

Building defense mechanisms within network nodes themselves will minimise the response time between attack detection and counteraction. Automating the defense mechanism will minimise the need for expert human assistance, which is in chronic short supply. Overall, the network will be able to dynamically respond to a wider range of threats and attacks.

The IBAN [32] and FIDRAN [33] models presented mechanisms to support vulnerability scanning and blocking using mobile code. In these models, mobile code based scanners are installed at network nodes which continuously moni-

[7] http://codered.newstrove.com/

tor for attacks based on supplied signatures. On detecting attack patterns, the scanner loads intrusion blocker services that suppress the traffic from the source of the attack. The additional feature of the FIDRAN model is that it is able to support packet monitoring services at the kernel level, thus improving system performance.

Van [34] demonstrated the use of active networks for defending against address spoofing. Van solved the address spoofing problem by dynamically deploying a filter which inspects all packets destined for a given host. Furthermore, by being able to autonomously push the filter towards the source of the attack, congestion that arises due to the attack is prevented.

Caching

Caching of objects within the network, rather than at the edge nodes, can greatly reduce network traffic and the time required to retrieve the relevant information. Traditional approaches to caching place large caches at specific points in the network. The key decisions to be made here are: (i) how to locate the required objects; and (ii) how to forward requests between various cache nodes in the network.

Bhattacharjee et al. [35] proposed that by associating small caches to network nodes, average round-trip latencies experienced by clients to obtain popular documents can be minimised. In this scheme, path segments between clients and the server nodes are partitioned into virtual groups of given radius, so that an item is cached only once inside a particular group. In addition to this, caches in a group also maintain pointers to objects that are present in caches of the neighbouring groups. These pointers are then used to reroute requests to the cache containing the object.

Legedza and Guttag [36] proposed a router level service to support cache routing. The major goal of their approach was to reduce the time required to find both

infrequently accessed, but cached, documents and those documents that are not cached. In this scheme, servers place per-document pointers at routers which aid in redirecting requests for popular documents to nearby caches. The result is that requests for infrequently accessed documents are allowed to travel quickly to the home server while requests for cached documents are redirected to caches present on the route to the server. This approach exhibits the benefits of associating pointers to cache servers within the network. Furthermore, individual caches do not need to know the addresses of neighbouring caches in the network in order to co-operate.

Data Transcoding

When operating over a heterogeneous network environment such as that of the Internet, packet flows that are customised for certain link characteristics (e.g. specific packet size and transmission rate) may be inappropriate for other links. The ability to change flow properties within the network will allow them to adapt to varying network conditions and to the requirements of end user nodes. Furthermore, flows with different characteristics can coexist; this will avoid the need to generate multiple flows at different rates when serving users with different terminal characteristics.

Sudame *et al.* [37] used pairs of active nodes to build flow transformation tunnels within the network. Active services attached at the entry and exit points of the tunnel provide application transparent route specific adaptation for flows when localised changes in network conditions are detected. Building such support within the network itself avoids the need for the traffic source to adapt to the single "Lowest Common Denominator" service available.

Amir *et al.* [38] used media gateways to perform transcoding of media streams within the network. The key feature of this approach is that individual users are able to instantiate static service modules at traffic gateway nodes to customise

media streams so as to suit end terminal characteristics and personal preferences. In the Application Level Active Networks (ALAN) project [39], mobile code was employed instead of static services to support media transcoding functions at active server farms within the network.

The media gateway nodes may become points of implosion under severe loading conditions. To overcome such problems, the Journey model [40] encoded application data units as independent media units that included customisation and computation parameters that can be applied on the unit by network nodes along the path. The primary benefit of this model is that it is able to distribute the processing load across multiple network nodes. Such scalability in design allows a large number of clients with different service requirements to be processed.

Congestion Control

Congestion is an intranetwork event, usually far removed from the application. The current Internet follows an end-to-end feedback-based approach for congestion control. In this approach the time taken for congestion notification information to propagate back to the sender limits the speed with which an application can self-regulate to reduce congestion. As a result, either there is a protracted period of time during which congestion is present, since applications have not yet learned about it, or else the notification arrives so late that there is no longer any congestion present and self-regulation is not required.

With active router support in the network it is possible to move the congestion control points into the network itself. The advantage of adopting such an approach is that the system can react faster to congestion, leading to lower packet loss which, in turn implies lower average delay and higher network utilisation.

Faber [41] presented the Active Congestion Control (ACC) scheme for congestion control in TCP. In this model, every packet includes the current window size adapted by the end application for transmission. When an active router no-

tices congestion, it calculates the new window size of the TCP packet and subsequently sends this information to the traffic source. Furthermore, the router drops those packets that would not have been sent with this new window size, thus resolving congestion immediately. As noted by Faber [41], nodes beyond the congestion point see traffic which looks as if the sender had reacted instantly. ACC is particularly effective, where the bandwidth delay product is large.

Bhattercharjee *et al.* [42] described an approach to application-specific congestion control within the network. Data packets are embedded with an application-specific congestion control service code. This service is triggered when congestion occurs within the network. This approach offers complete control under congestion conditions to individual data packets themselves, thus offering maximum flexibility in operation.

Random Early Detection (RED) [43] mechanisms are widely accepted for congestion avoidance in the Internet. However, it has been shown that when employing RED mechanisms, unresponsive bandwidth greedy connections get a larger than fair share of the bandwidth at a bottleneck link when competing with responsive connections [44]. Niraja *et al.* [45] presented a solution using mobile code-based packet filters to control bandwidth greedy connections from hijacking node resources. When a node notices congestion, it finds the set of connections that are bandwidth greedy and deploys filters in the network to drop packets belonging to such flows. These filters are progressively relayed by the active nodes towards the source of the greedy connections, so that packets drops are made early. This will allow a saving of network resources and would also support fair sharing of the network bandwidth.

A sender initiated approach to support QoS routing in the Internet has been presented in [46]. In this model, when a QoS supporting path is required, the source sends an active packet towards the traffic destination which contains the QoS parameters that the path should satisfy. The active packet in conjunction

with the intermediate active nodes, gathers the required path state information, evaluates and reports back the specifics of the best available path to the node that deployed it.

Multicasting

Active nodes were used in [47, 48] to suppress duplicate NACKs from reaching the traffic source, thus avoiding the NACK implosion problem. In [47], by supporting soft-state information at active nodes, repair packets were only delivered to those receivers experiencing data losses. This reduces redundant traffic within the network and also minimises the burden on end receivers. Furthermore, by supporting best-effort data caching within the network, the end-to-end error recovery delay was also minimised.

In [49], Yamatoo *et al.* described an approach to multicast communication support to unicast hosts using active network services. The active services were deployed on demand to act as gateway points between the unicast hosts and multicast enabled network segments. Active services acting as reflectors duplicate packets from the multicast source to individual unicast hosts. The important contribution of this model is that active services are able to maintain application level connectivity even in the case of multicast failures.

Another problem with existing multicast protocols is that their design does not include support for heterogeneous group communication. Existing IP multicast protocol models serve heterogeneous receivers either by: (i) using a single rate flow to all receiver nodes irrespective of end terminal characteristics and the network path segments connecting them; or (ii) generating multiple flows with different rates over different multicast trees and, allowing the receivers to subscribe to an appropriate delivery tree depending on their preferences and constraints [50]. The first scheme has been found inadequate to serve heterogeneous user groups [51], while the second scheme employs resources inefficiently.

In contrast, with packet processing support, nodes can perform client-specific adaptation of data within the network based on user requirements and available network resources. Furthermore, such schemes avoid the need for receivers to continuously probe and change subscription levels so as to adapt to network congestion.

In [52], receivers with similar service demands were logically grouped into distinct multicast receiver groups, referred to as service level groups. Each such group was then served by a local active Node which worked to adapt the original data according to the local user requirements. Keller *et al.* [53] used kernel level network services referred to as router-plugins to dynamically adapt video streams to momentary load conditions at network nodes. This scheme provided conservation in resource usage by allowing each branch of the tree to adapt to its maximum available throughput.

2.5 Active Networks: Themes and Concepts

Mobile Code: The term mobile code describes any program that can be shipped unchanged to a heterogeneous collection of processors and executed with identical semantics on each processor [54]. The underlying feature of such a paradigm is that software modules are able to dynamically change the bindings between the code fragments and the location where they are executed.

Developing applications as mobile component allows network nodes to dynamically load the required application-specific service code according to demand, thus avoiding the need for permanent storage resources. Downloading and executing mobile code is an established technique for supporting Java applets in the Internet. Similarly, active networks employ this concept to dynamically provision new networking services on demand.

Active Network Node: The core entity of any active networking architecture is the active node, which provides support for dynamic service provisioning. In

active networking terminology this model is commonly referred to as the "active node" (Fig. 2.4). The three major components of the node are: (i) the Execution Environment; (ii) the Node Operating System (NodeOS); and (iii) the Network Services (NS). The active node architecture and nomenclature emerged from DARPA's active network community [55]. Their architecture is shared by most of the existing active network proposals.

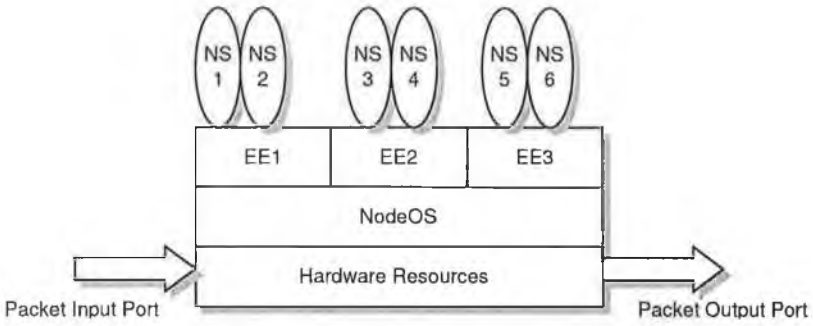


Figure 2.4: Architecture of an active node

Cross-section of an Active Node

The Execution Environment: The execution environment is central to an active node. It can be a virtual machine or a set of rules that provides support for the execution of services at a node. The use of a virtual machine representation of network nodes provides a common logical model of the hardware, thus overcoming the problem of hardware heterogeneity.

The fundamental responsibilities of an Execution Environment include: (i) to interpret incoming packets and pass them to corresponding network services for processing; (ii) to support dynamic installation and removal of network services; and (iii) to manage node resources and access rights among the network services on the node, thus providing a secured framework for service provisioning. Multiple Execution Environments can be supported by a node (Fig. 2.4).

The Node Operating System (NodeOS): The NodeOS forms a shim layer be-

tween the Execution Environments and the node resources (e.g. memory, processor cycles). The NodeOS defines four programmable node abstractions: threads, memory, channels and flows. The first three entities abstract the computation, storage and communication capacity used by Execution Environments. The flow entity abstracts user data-paths with security, authentication and admission control facilities. The NodeOS interface gives each Execution Environment fair and controlled access to both the computation and the transmission resources of the network node.

Network Services (NS): Network services are programs that implement the required application service-logic within the network using the programming interface supplied by the execution environment. The mechanism by which these network services are loaded/removed from the networks nodes is dependent on the active network model followed (see section 2.6). In this thesis, the terms network services and network protocols are used interchangeably.

Active Network Encapsulation Protocol: The Active Network Encapsulation Protocol (ANEP) [56] has been proposed for encapsulating active packets over different lower-level media and protocols. This protocol provides a common packet encapsulation format over a network in which different Execution Environments may coexist. Specifically, ANEP encapsulates the active payload (i.e, the network service or a reference to the service) with a header that contains a version number, a flag field, an identifier that indicates the active processing engine and information on the packet length details.

2.6 Active Networking Models

Active Packet Model: In this approach, packet handling services are integrated into every packet of data sent into the network. When such packets arrive at an active node, the EE interprets the program in the active packet and makes the forwarding decision for the packet based on the program execution results. This

model is also commonly referred to as the "in-band" active network approach.

Active Node Model: In this approach, network services are injected separately from the actual data packets. Active network models using this approach typically use packets that carry some identifiers or references to predefined functions that reside in the active nodes. If the requested service is not available at a node, it is dynamically downloaded from code servers located at convenient points in the network. This model is commonly referred to as the "out-of-band" active network approach.

2.6.1 Active Packet Models

Active IP [57]: The Active IP project demonstrated the feasibility of building an active network within the IP protocol. The major applications of this model was to solve tasks related to network probing and discovery. Packets carry miniature programs (coded in Tool Command Language or TCL) that are executed at network nodes. The processing engine at the active node i.e. a TCL interpreter, is positioned adjacent to the IP layer, and is invoked when the packet passes through the layer. One of the major drawbacks of this approach is that the size of the embedded program fragment is limited by the size of the IP options field. Furthermore, this model does not address security issues except for validation control, as offered by TCL.

Smart Packets [58]: BBN's Smart Packets project demonstrated the use of active networks for network management and monitoring purposes via SNMP like interfaces. Smart packets are used to configure and react to alarms from network nodes thus reducing management traffic and removing the requirement for a centralised management system. Packets in this architecture contain service code in Spanner, which is a RISC-style assembly language. These packets are designed to be self-contained, thus requiring no state storage within the network nodes. Therefore, embedded programs have to be smaller than the MTU size so as to fit

into a single link layer packet.

In comparison to the Active IP model [57], this approach only addresses network management tasks and does not support the introduction of new network services. In this model, extending services dynamically is not feasible as this would require modifying the virtual machine itself. However, this approach specifically addresses security concerns by adapting a heavyweight cryptography mechanism to authenticate access control and to check data integrity of the incoming active packets.

Safe and Nimble Active Packets (SNAP) [59]: An approach based on formal methods was followed in SNAP to add in a higher level of safety when compared to other Active Packet models. SNAP is a stack-based active networking language, based on the Packet Language for Active Networks (PLAN) [60]. PLAN programs are strongly typed and statically type-checked to provide a secure working environment. By restricting the PLAN language to guarantee that all programs are safe SNAP achieves control over resource usage at network nodes. SNAP based active packets employ bandwidth and memory in linear proportion to the packet's length. Hence a node can calculate an upper limit on the resource usage of a packet operating at a network node.

SNAP has been demonstrated to solve tasks mainly in the field of distributed network management. The evolution and adaptation of SNAP depends on the PLAN language itself. Due to the restricted nature of the PLAN language and the small population of programmers with expertise therein the scope for network innovation is limited.

M0 [61]: The M0 architecture uses the term messenger for its Active Packets and correspondingly active nodes in the M0 model are referred to as messenger nodes. Each messenger includes service code that can be applied to its own data field. Embedded code in messengers are written in M0, a stack based language similar to Postscript. The interpreter for M0 code is written in C.

Messengers at active nodes are executed by an independent thread of control. Furthermore, each messenger is allotted private memory space and shielded from other messengers operating in parallel at the same network node. For realising a complex service that requires a large code base, messengers implement their own caching method by storing the code in a shared memory area of the node under a chosen name. M0 adapts a credit trading mechanism to support resource sharing among simultaneously operating messengers. The major problem with this approach is that it implements network services on a per-flow basis which is not scalable for large networks such as the Internet.

2.6.2 Active Node Models

Active network Transport System (ANTS) [62, 63]: ANTS from MIT was one of the pioneer models that introduced and used the concept of capsules for building and dynamically deploying network protocols. The capsules (analogues of packets in the current network model) include references to the network services that must be used to process them at each active node. A novel approach referred to as "load-from-source" was proposed to deploy network services at intermediate network nodes. On the arrival of a capsule at an active node, the local protocol cache is checked. If required code is not present in the cache, the capsule is put to sleep for a finite time and a load request for the missing portion of the protocol is sent upstream towards the source of the packet. The prototype of ANTS is implemented in Java. Security in ANTS is enforced through the use of the Java sandboxing environment. Furthermore, MD5 [64] is used to digitally sign each capsule to prevent capsule spoofing.

ANTS supports demand downloading and caching of the routines at active nodes on a per flow basis. One of the novel features of this approach is that of the code caching at network nodes to improve on service deployment latency. However, the "load from source" approach is simple but may not be the most efficient

strategy. A service discovery protocol, which has good dynamics is presented in section 3.3.2.

Practical Active Network (PAN) [65]: PAN is a variant of the ANTS model [62]. Capsules transfer binary code and execute them directly, trading performance for security. Its eventual goal is to provide performance comparable to existing passive networks while providing a safe execution environment for mobile code. A unique feature of PAN is its support of multiple code systems, including native Intel x86 object code and Java.

By loading binary objects directly into the kernel, PAN avoids the kernel crossing overheads suffered by most virtual machine based active network systems. The authors claim that the most significant bottleneck is the performance of mobile code systems such as the Java virtual machine, which involves data copying, and extra context switches as well as the overhead involved in using a general purpose virtual machine. The performance improvement in the PAN model is achieved at the expense of security.

ASP EE [66]: The ASP EE is a Java-based active network Execution Environment. It essentially acts as a mini-operating system in which ASP network services (AAs) can be dynamically loaded and executed. An important feature of ASP is the support of persistent active services that may have long-lived execution threads.

TAMANOIR [67]: The TAMANOIR model is another variant of the ANTS [62] model. It was able to achieve a performance improvement of a factor of two over the standard ANTS model when tested for forwarding IP datagrams. This was achieved by employing a Java compiler instead of a JVM. The code compiler used for this purpose is the GNU Java compiler[8].

PANDA [68]: The PANDA architecture, also based on the ANTS [62] model, aims to integrate active network support to legacy applications. This aspect will be of

[8]http://gcc.gnu.org/java/

importance in encouraging roll-out of active networking systems. PANDA performs conversion of legacy datagram packets to active packets that either include code or reference to service code which enhances end application functionality when working under different network conditions. This architecture has demonstrated active support for stream based applications such as that of HTTP and POP.

Distributed code Caching for Active Networks (DAN) [69]: The DAN approach differs from the ANTS model in two principal aspects: (i) this model uses native code (x86 object code) rather than mobile code; and (ii) the mechanism by which service code is deployed into the network is different.

DAN employs distributed code servers to house service code within the network. Code servers feature a database of network services for a range of operating systems and hardware architectures. When capsules (i.e. active packets) arrive requesting services that are unavailable at the node, the required service code is downloaded from an authorised code server. Security concerns are addressed by using well known code servers which authenticate themselves. Furthermore, code modules are expected to be digitally signed so as to avoid malicious service code.

AMNet [70]: This active model advocates the use of native code based active services as in the DAN approach. The current implementation features a Linux-based OS on which the AMNet EE is present. AMNet allows native code to be dynamically loaded into the kernel, which has serious security implications. The AMNet model proposes to use a wrapper layer between the service module and the node's operating system, so as to ensure system integrity. In AMNet, code servers are arranged in a trusted hierarchy similar to the DNS nodes. By establishing mutual trust between the code servers, a global secured framework of distributed code servers can be built.

Composable active network Elements (CANES) [71]: CANES adapted concepts

from advanced intelligent networking [72] to the Internet, it offers only a predefined set of network services through an in-built API. Thus, unlike other architectures this model does not support dynamic service provisioning. Users in this architecture are only allowed to define the set of services (through references present in the packet) that will operate on their data stream rather than introducing them dynamically. The available services are chosen and implemented manually, for example by the network administrator. This model achieves better performance and has fewer security concerns than other active network models, at the expense of flexibility.

Switchware [73]: The Switchware architecture features a network model which is a hybrid of the Active Packet and the Active Node models. Active packets in Switchware are programmed in PLAN [60]. PLAN programs are made compact and secure by deliberately restricting their actions (e.g. a PLAN program cannot manipulate node resident state). However, to realise complex network services, PLAN programs embedded in Active Packets refer to active extensions called Switchlets, which are dynamically loadable service modules written in CAML[9]. CAML offers formal methodologies to prove the security properties of the Switchlet modules at compile time and no interpretation is required. The architecture only supports explicit code loading rather than the on-demand loading of models such as ANTS [62] and DAN [69].

NetScript [74]: Netscript is a programming language and environment for building networked systems. It is a dynamic dataflow language based on object oriented concepts. The Netscript model views the network as a single programmable entity rather than a collection of heterogeneous network nodes. Based on this abstraction, a set of distributed network nodes in Netscript are collectively referred to as a Virtual Network Engine (VNE). VNEs provide an abstraction of network resources that can be programmed and managed as a single object. The Netscript

[9]http://pauillac.inria.fr/caml/index-eng.html

model is intended to provide support for control plane tasks rather than data plane processing.

DARWIN [75]: The DARWIN model isolates itself from the data path programmability and restricts itself to programming the functions of the control plane. Network services referred to as delegates are deployed by service providers to configure the control plane functionality of network nodes. The node resources are organised into a hierarchy for management purposes. Defined delegate operations include data merging, flow splitting and changing routing functionality. The ability to dynamically customise network resources on-the-fly per flow is a novel feature of this architecture. Although the routing and programmable entities reside on the same node, the programmable component does not interfere with normal fast path packet forwarding.

Active Services (AS) [38]: This project advocates the placement of computational nodes, referred to as active proxies, at strategic locations within the network to support user customised data processing. In this model, users instantiate static service agents, referred to as servents, at one or more locations to perform customised multimedia delivery over a best-effort Internet. The active services are written in MASH [76], a platform based on Tcl but extended and optimised for multimedia operations. The AS framework does not address the broader issues of dynamic protocol deployment. A similar framework to the AS model, referred to as Xenoserver, was presented in [77]. The additional feature of this approach is that it presents an environment in which resource usage is strictly scheduled, accounted and charged for.

Application Level Active Networks (ALAN) [39]: In this architecture, users upload service modules, referred to as proxylets, to distributed dynamic proxy servers within the network to perform custom handling of their data streams. The major difference between the ALAN and the AS [38] model is that the former uses mobile code for service provisioning, while the latter uses static code for network

services. Unlike typical active networking schemes, which intercept packets directly on the forwarding path (network layer), ALAN requires the data streams to be explicitly addressed to the proxy servers. Furthermore this approach is not very practical since active services cannot be applied in a transparent manner to the end systems, since end applications or end-systems must address packets explicitly to a proxy active node.

Active Engine (AE) [78]: The goal of this model is to build an efficient distributed network management framework. This model follows the Active Packets model to transfer service code to active nodes. In this architecture, an active node comprises of a pair of network elements: (i) the traditional high speed IP router; and (ii) Active Engine - typically an execution environment where network services are executed. Packets that request additional processing are diverted to the AE using a packet classifier operating at the IP router. SNMP [79] agents are employed to communicate between an AE and its corresponding routing element. This model demonstrates the feasibility of integrating active network functionality into current networks incrementally. A similar effort to that of AE is being carried out by the SARA group [80] with the goal of building an architecture that supports a wide range of tasks, of which network management is one.

2.6.3 Active Packets vs. Active Node Models

Flexibility: Flexibility refers to the possible range of network services that can be defined using an active network model and the degree of network customisation support available for third-parties. In the Active Packet approach, the maximum possible size of a service code that can be carried in an individual packet is restricted by the MTU of the network path segment[10]. Due to this limitation, Active Packet approaches [57, 58] are only employed for simple network management tasks (e.g. to gather the status of the output buffers of a set of network

[10]Note on ethernet LANs the MTU size is 1500 bytes.

nodes) rather for providing new network services. However, the M0 model [61] by employing a caching scheme builds a more flexible framework compared to other Active Packet models [57, 58]. A major problem with the M0 approach is that it implements network services on a per-flow basis which is not scalable for large networks.

The Active Node model achieves increased flexibility when compared to the Active Packet approach. The major reason for this is due to the use of an out-of-band code loading scheme. The Active Packet approaches do not limit the third-party programmability to any predefined set of user groups. In comparison, all available Active Node models do not support the same level of flexibility.

Active Node models [65, 81, 82] (section 2.6.2) employ native code-based services rather than using platform neutral service code, to achieve improved performance levels, at the expense of flexibility. However, security threats are considerable when users are allowed to load native code-based services directly into the kernel. In order to build a secure environment, service provisioning in these models must be restricted to a set of third parties who are trusted by the network provider.

Scalability: The Active Packet approach requires individual packets to be processed by the accompanying code at intermediate network nodes. Performing per-flow packet processing operation inside the network affects the network scalability. This is because, each individual active packet will require memory for storing the relevant service code and also CPU cycles for processing. Thus, as the number of simultaneously operating sessions increases, there is a linear increase in memory requirements and packet processing delays. Given the amount of data flowing through the the network, any approach which requires per-packet storage of both code and data is likely to prove impractical.

By contrast, in the Active Node model, it is possible to associate a single service code to a group of flows rather than to individual packets. This allows the

cost of deploying service to be amortised over the set of flows, thus making the model scale to large networks and user populations.

Efficiency: Active network systems employ bytecode representations of network services to provide architectural neutrality and code compactness. However, the need to perform code interpretation using virtual machines results in poor execution speeds [83]. Dan Decasper *et al* [84] argue that – "in order to route at a rate of 10Gbps, a computer running at 300MHz has only 234 cycles to receive, process and forward a packet of 1KB".

This precludes the possibility of using currently available virtual machines and interpreters for high-speed active networking. This bottleneck in the available processing power is expected to persist in the foreseeable future, since transmission speed is growing at 200% per year, out-pacing the growth of processing power, which still follows the Moores law [85]. This is a major reason why capsule based active network systems such as ANTS [62], which require every packet to be processed, are not suitable for high-speed networking.

Tests performed on active network models such as PAN [65] and DAN [69] which work with native binary code have shown that the performance penalty is not inherent to the active network architecture, but is due to the limited available processing power of byte-code interpreters. The availability of high speed byte-code processors will improve performance. The TAMANOIR model [67] has demonstrated that, by using byte-code compilers instead of interpreters, the performance of active nodes can be improved.

2.7 Mobile Agents

The majority of existing active network systems employ mobile code to support the runtime extension of services at network nodes. Alternatively, mobile agents can be used to realise active networking systems. Before discussing the need for

and benefits of building such an architecture, we take a brief look at the mobile agent paradigm and its origin.

"The idea of an agent originated with John McCarthy in the mid-1950s, and the term was coined by Oliver G. Selfridge a few years later, when they were both at the Massachusetts Institute of Technology. They had in view a system that, when given a goal, could carry out the details of the appropriate computer operations and could ask for and receive advice, offered in human terms, when it was stuck. An agent would be a soft robot living and doing its business within the computers world" [86].

The term agent emerged from the field of Artificial Intelligence. However, this concept has been successfully adapted to the area of distributed computing.

The Artificial Intelligence (AI) community define agents as – *computer programs that simulate a human relationship, by doing something that another person could otherwise do for you* [87].

Researchers from the field of distributed computing define agents as software components that act alone or in communities on behalf of an entity and which are delegated to perform tasks under some constraints or action plan [6]. It is in this sense that the term "mobile agent" is used in this thesis.

Though there is little consensus among researchers about what an agent is, it must be:

- autonomous and asynchronous - *it has control over its actions*;
- reactive - *it senses and adapts to changes in the environment*; and
- goal oriented - *it works towards accomplishing the action plan.*

Additionally, an agent may possess one or more of the following attributes, depending on the nature of the task to be accomplished. It may be:

collaborative - *it can work in concert with other agents to achieve a common goal*;

mobile - *being able to migrate between hosts in a network in an autonomous manner*; and

able to learn - *it adapts in accordance with previous experience*

A mobile agent is an active program that acts on behalf of a user or another program but under its own control. That is, the agent can choose when and where to migrate in the network and can decide on how to continue its execution thereafter. The difference between mobile code and a mobile agent lies in the inclusion of states. In general, the term "state" refers to those attributes of an agent, which help to determine its behaviour when it resumes execution at a new location.

A mobile agent is composed of the code describing its behaviour and data and execution states that are associated with it. The term "code," refers to the classes (in the sense of object-oriented programming) necessary for the agent to execute in the new location. The term "execution state" refers to the stack and program counter values of the migrating entity. This state attribute allows the agent to continue from the point where it was stopped before migration. Finally, the term "data state" refers to the values of the internal variables of the migrating entity.

2.7.1 Mobile Agent Systems

A number of implementations of agent systems eixst. These systems can be categorised based on the programming languages for which they provide support. They either support: (i) agents written only in one language – *single language systems*; or (ii) agents written in multiple languages – *multiple language systems*. Some contemporary Mobile Agent (MA) systems that fall into the former class include IBM Aglets [88], Mole [89], Odyssey [90], Grasshopper [91], Concordia [92], Telescript [93], Obliq [94], Ajanta [95] and AgentBuilder [96]. Systems of the latter kind include ARA [97], D'Agents [98] and Tacoma [99]. A comparison of a variety of mobile agent platforms can be found in [100].

A noticeable feature of these agent systems is that they have the same general architecture: a server running on each host accepts incoming agents, and for each agent, starts a process/thread, loads the agent's state, and resumes agent execution. Furthermore, the language used in all implementations are interpreted or scripting languages (e.g. Java, Tcl). Even where a "traditional" language such as C/C++ is used, they are also essentially interpreted [97,98].

Agent Mobility

The mobility characteristics of an agent is determined by the composition of execution and data states. Agent mobility can be categorised into two types [101]: (i) strong mobility; and (ii) weak mobility.

Strong Mobility: This mode of mobility involves transporting both the execution and data states with the code to the new location. This will allow the agent to continue execution from the exact point at which it stopped before migration. For example, when an agent migrates under this model all objects and resources and all threads created by the migrating entity are also transferred to the new location. This model may seem similar to the case of process migration in distributed operating systems. However, the major distinguishing feature is that, in process migration the underlying system is in control of distributing the processes, while agent migration is self-directed.

Weak Mobility: In this model only the data state is transferred with the code to the new location. The execution of the agent at the new location starts from a specified procedure (a method in the case of objects) as configured by the agent programmer. Furthermore, the amount of the data state information that will be packaged for transfer can be specified by the agent programmer. This feature allows the size of the agent to be controlled.

Safety and Security

The flexibility to inject third-party code into networks introduces a wide range of safety and security problems. The threats related to the use of mobile code can be categorised into two classes:

- Threats faced by an Agent Node: The node should be protected from both malicious and erroneous code. Mechanisms are essential to protect node resources, and prevent agent programs from disrupting other agents in the system or even locking up the whole system due to resource misuse.
- Threats faced by a Mobile Agent: Mobile agents execute tasks in third-party nodes on behalf of users. Hence, it is important to protect the code from being tampered with, to ensure its confidentiality when necessary, and to guarantee its integrity.

There is a considerable amount of ongoing research efforts to address the first category of threats. The most widely adopted approaches are authentication and sandboxes.

Authentication provides a means to validate an agents identity. Authentication is commonly achieved through cryptographic means [102]. These methods normally require the consultation of some form of public key service or certifying agent to verify the cryptographic protection (i.e. digital signature or private key) [103]. Common cryptographic algorithms for authentication include public key signatures, keyed hashes (e.g. MD5 [64]).

Software-based fault isolation method isolates application modules into distinct fault domains, enforced by software [104]. This technique is also referred to as sandboxing. The idea here is that untrusted code will only be allowed to operate on a predefined region of memory. The code is then instrumented to be sure that each load, store, jump instruction is to an address, which is in the fault domain, which is assigned to the code. This ensures that the code cannot break

out of the safe processing environment. Malicious programs trying to exceed the sandbox boundaries are typically stopped and removed.

A third technique, referred to as Proof Carrying Code (PCC), is also widely proposed to provide safety mechanisms in agent systems. PCC is a technique to support program verification [105]. PCC associates a formal proof of certain safety properties with a compiled program. In this, a code receiver establishes a set of safety rules that guarantees safe behaviour of programs in the local environment. The code producer creates a formal safety proof[11] that proves adherence to safety rules for the untrusted code. Based on this, the receiver will be able to check whether the untrusted code is deemed safe for operation at the node.

Mechanisms to protect mobile agents against attacks from malicious hosts are considered to be one of the most difficult security problems. Yee *et al.* [106] introduced the idea of a "Sanctuary" as a closed tamper-proof hardware subsystem where agents can be executed in a secure way. In this model, the trust is moved from the agent system to the manufacturer of the hardware. Tschudin *et al.* [107] proposed a model based on encryption functions. In this approach, mobile agents are executed directly as encrypted programs at agent destinations. Finally, a decryption function is used when the agent reaches the source host to recover the results. Research in this area is still in its infancy, and it is not yet clear as of now how well these solutions will perform.

2.7.2 Mobile Agents for Networking Applications

Traditionally, mobile agents have been advocated for use in end user applications, such as personal assistants for information retrieval or as shopping agents in electronic markets. Recently, a wide variety of networking applications have also demonstrated the benefits of using mobile agents. Some of the key areas includes:

[11] To compute the safety predicate for a program means to encode the semantic meaning of the program in logical form and constitutes a formal statement that the program, when executed, will not violate any safety checks [105].

(i) network management [108], (ii) distributed resource management [109], (iii) dynamic network routing [110], (iv) distributed monitoring [111], (v) resource discovery in distributed environments [112] and (vi) ad-hoc networks [113]. The advantages of using mobile agents emerge from some of its inherent properties. They are:

- Agents can move to the place where the data is stored, realising queries and filtering relevant information before sending the data to the client.
- Mobile agents are independent from the hosts that were responsible for launching them into the network. Furthermore, they are capable of continuing to work even if the delegating entity does not remain active.
- The autonomous nature of mobile agents allows systems to be built that can dynamically adapt to circumstances.

2.7.3 Mobile Agents and Active Networking

The mobile agent paradigm has also been advocated for realising active networking systems [7, 114–116]. Though the goals that led to these two paradigms vary significantly, the features of the mobile agent paradigm can be exploited to realise an active network architecture. The computational entity (i.e. code/service logic) that an agent transfers can be considered as a "network service" in an active network environment. By exploiting the concept of code mobility of the mobile agent paradigm, it will be possible to distribute network services to multiple points in the network. These distributed services can then be used to control the movement of data within the network. Furthermore, individual mobile agents themselves can be deployed to function as a network service for packet processing. The fundamental requirement to build such a network model is that agent execution environments will have to be present within the network cloud, rather than at its periphery. Hence, we can view the mobile agent paradigm as a means

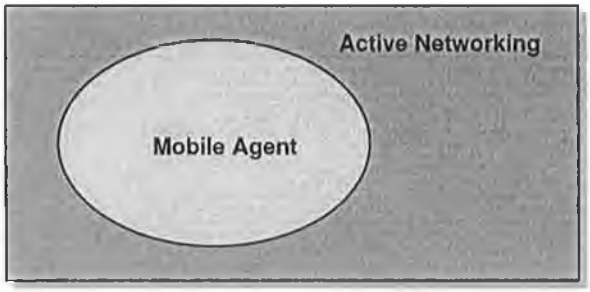


Figure 2.5: Active Networking and the Mobile Agent Paradigm

to build an active networking architecture (see Fig. 2.5).

2.7.4 Why use Mobile Agents for Active Networking?

Employing an agent based active networking system provides a flexible framework to build a variety of new networking services. Furthermore, it provides a unified framework for service deployment and network management purposes. Below we expand on these claims.

New Classes of Network Services

Self-organising Services: Network services based on agents can self-organise and co-operate within the network to accomplish user specified tasks in an independent manner. Such a feature allows networks to be built, which will be able to automatically identify and respond to events in an independent manner. We propose a practical example of such a service in Chapter 4; this service autonomously identifies and manages non-QoS islands in QoS networks.

Time and Location Specific Network Services: By employing mobile agent based networking services, users can be provided with autonomous services that represent them at distributed points within the network for predefined time periods. These services will be able to initiate actions and accomplish goal directed behaviours on behalf of the user. We propose two practical examples of such

services in Chapter 4; they include services to provide support for content distribution networks and large scale multicasting in the Internet.

Reduced Management Complexities

Existing active networking systems only provide static networking services, i.e. once services are deployed and activated they cannot take into account the state of the node on which they operate (e.g. CPU load, link bandwidth). End users are also required to manually perform reconfiguration or relocation of the service when the operating conditions of the active nodes changes. Such a model can jeopardise the complete networking system due to excessive control and management traffic. Employing a model which requires continuous manual intervention for service management or operation will not scale as the population of network services increases.

In contrast, making the network services automatically adapt (based on in-built intelligence) to the state of the operating node will avoid the need for manual service management. Realising such a system reduces the manual control and management tasks required to support proper operation. Overall, such a model scales to large networks and user populations. Such a model takes us a step closer to the concept of proactive computing [117], which visualises a migration from human-centred to (un)supervised computing.

The Combined Active Network Model

The mobile agent based active model can be used to realise both the in-band and out-band active network approaches under a single framework. Furthermore, limitations of the in-band approach (section 2.6.2) do not appear in this model.

Commercial Acceptance

Another advantage of employing an agent based solution is that emergence of commercial active networking systems can be fast. The major reasons for such a claim are: (i) currently, there are many varieties of mobile agents systems available for commercial purposes; adopting them to suit active networking would be relatively easier, than convincing vendors to implement new active networking systems; and (ii) the mobile agent community is large, hence experience and expertise gained from implementations of agent systems can be utilised.

2.7.5 Why Netlets?

The Netlets architecture [7] uses mobile agents for building an active networking architecture. Netlets are autonomous, nomadic mobile components which persist and roam in the network independently, providing predefined network services. Research efforts, similar in flavour to the Netlets model, have been presented [114–116]. However, the design goals that are unique to Netlets are discussed below.

Distributed Architecture: The Netlets architecture follows a decentralised approach for code delivery. Every node on the network participates in the hosting of host Netlet services. Such an architecture overcomes the problems faced with centralised and hierarchical service distribution schemes of existing active networking systems (see section 3.1.2).

Demand based: The Netlets concept advocates a demand driven protocol model based on a biological metaphor. In this, the life of a particular type of Netlet would be purely based on the user demand for that service. On an increase in demand for a particular type of Netlets, the required set of services are automatically replicated and dispersed into the network, while the obsolete or the

unsuccessful services are quietly deleted. This leads to a demand-driven population of services in the network. The key motivation that led to such argument is as follows. Active networks advocates third party services to be deployed in the network. Manual management of services in wide area networks would not scale. In contrast, a model based on the biological metaphor would allow the network to evolve in response to user demand.

The Need to Develop the Netlet Execution Environment

Currently, there exist a wide variety of mobile agent systems (see section 2.7.1), which can be used to develop the execution environment for Netlet operation. One of the major limitation that prevents us from employing existing agent systems to build Netlets, is that they are monolithic and heavyweight.

Currently available mobile agent systems are intended to support user applications, such as software robots for shopping in e-markets, personal assistant agents, etc. A wealth of features is required to support such diverse applications. Thus, the basic mobility framework is augmented on with value-added services which are not required in the case of active networking architecture, such as Netlets.

Existing agent systems are expected to run on end systems, such as web servers and user computers, where there are few constraints on processing power and memory. In contrast, Netlet-like active services are required to operate on resource constrained, network devices with hard real-time constraints.

Finally, the Netlet execution environment requires packet communication support to provide packet processing in the network. Existing agent systems, do not provide such support. The unique requirements of the architecture coupled with limitations of existing systems, have generated the need to build the Netlet prototype.

Selecting the Programming Language for Realising Netlets

The next step is to select the appropriate language that would be most suitable for developing the Netlets architecture. The key language features include support for:

- network communication;
- mobile code communication;
- dynamic code loading;

Also of interest are the safety and security features supported by the programming language. The programming language defines the range of operations third-party Netlets can perform on a network node. One of the important security related feature of a language is to provide strong typing support. With strong typing in place, third-party untrusted services will be restricted to predefined memory segments, thus avoiding common programming errors.

Another crucial feature is realtime performance. This is because Netlet services are expected to operate on resource constrained network devices, such as embedded processors. These devices operate to hard real-time constraints. Finally, the language should be widely used. Though, this is a non-technical feature, it will decide the popularity of the model. Furthermore, this will allow the system to evolve continuously with the language.

Java [12] supports the majority of the features listed above. The major shortcoming of Java is essentially its performance. Efforts are in progress to improve the realtime performance of the language. With the emergence of new technologies, such as Java-based processors [118], better performance will be obtained.

[12] http://wwww.java.sun.com

2.8 Summary

The need to rapidly develop and introduce new services has instigated the need to revolutionise the way networking systems are built. It has been identified that by replacing the closed vertically-integrated systems of today with programmable network nodes, it will be possible to realise a flexible networking architecture. Two major schools of thoughts have been proposed for the realisation of the programmable networking paradigm. They are (i) the Programmable Node approach; and (ii) the active networking approach. The active networking approach provides a more flexible network infrastructure than the Programmable Node approach. Hence, the former was taken up for further investigation.

Currently, there are two major flavours of active networking systems. They are: (i) the Active Packet model; and (ii) the Active Node model. The former advocates the idea of embedding service code within each packet, while the latter follows a code-on-demand approach for service provisioning. Alternatively, mobile agents can be used to build active networking architectures. Such an architecture provides a flexible framework for realising new classes of networking services, rather than the existing active models. Furthermore it provides a unified framework for service deployment and network management purposes.

The Netlets architecture follows this idea of using mobile agents to realise an active networking system. Netlets are autonomous, nomadic mobile components which persist and roam in the network independently, providing predefined network services. The Netlets architecture follows a decentralised approach for code delivery. Furthermore, it follows a demand driven model for code replication in the network. This thesis presents a case for introducing network programmability using Netlets, in way such that it can augment and support existing network protocols, thereby providing new networking services.

Chapter 3

Netlets Network

This chapter presents the Netlets architecture and the mechanisms used to distribute Netlet services in the network. First, I present the set of methods that I developed to support the execution of Netlet services at network nodes. I describe the implementation of the Netlet prototype. The remainder of this chapter concerns the reactive and proactive service deployment schemes used to distribute Netlet services in the network. Also I propose: (i) a DNS based distributed system, which allows Netlets to discover where active node support is available in the Internet; and (ii) a service discovery protocol, referred to as the Stigmergy, which supports the discovery of active services in the network.

3.1 The Netlets Network Architecture

Netlets are autonomous, nomadic mobile components which persist and roam in the network independently, providing predefined network services. Netlet nodes offer runtime environments for the operation of Netlet services. A simple IP based Netlets network is shown in Fig. 3.1. It consists of both Netlet nodes and regular IP network elements (e.g. routers, switches).

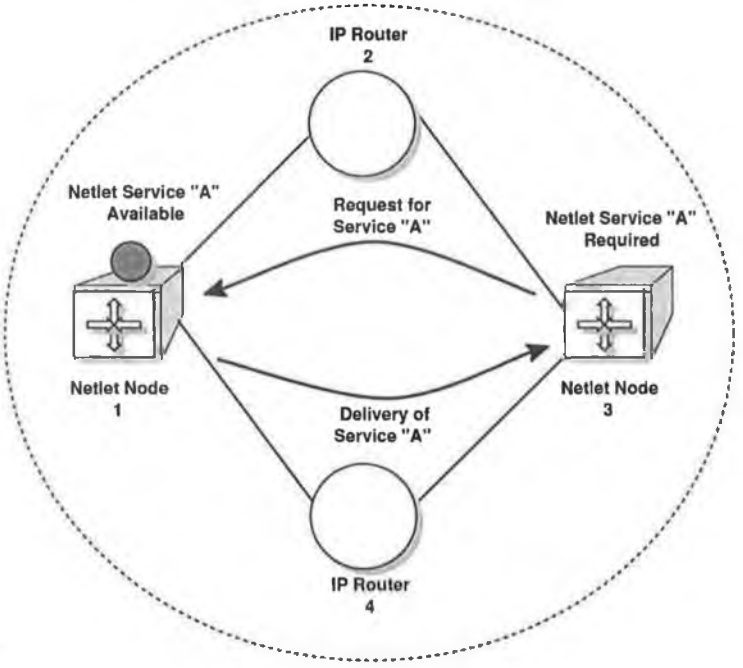


Figure 3.1: A Simple Netlet Network Model

3.1.1 Architecture of a Netlet Node

The general architecture of a Netlet node is shown in Fig. 3.2. It consists of: (i) a packet forwarding layer; and (ii) a packet processing layer. The former provides a high-speed path for forwarding regular data packets as in existing network elements, while the latter functions to support packet processing using Netlet services.

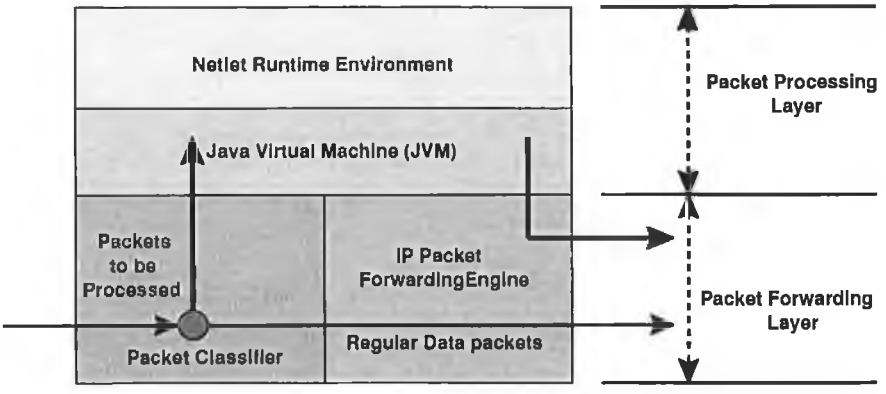


Figure 3.2: Architecture of a Netlet Node

At a Netlet node (see Fig. 3.2), packets are only processed by the Netlet layer when:

- there is an explicit request by packets themselves, (e.g. by setting router alert fields) – this scheme is referred to as **reactive packet processing** in Netlets; or
- at the instigation of Netlet services, (e.g. by Netlets setting up packet filter rules within the packet classifier) – this scheme is referred to as **proactive packet processing** in the Netlets architecture.

The main motivation for separating packet processing from that of regular packet forwarding functions is the following. The level of data traffic the network is required to transport is increasing with time. It is not possible to support packet processing speeds that match commercial grade high-speed packet forwarding techniques even with the state of the art in software technologies. Hence, performing selective packet processing at Netlet nodes will maximise the efficiency of the architecture.

Netlet Runtime Environment: The core of a Netlet Node is the Netlet Runtime Environment (NRE). The NRE provides the execution environment support for operation of Netlet services at a Netlet node. To integrate mobility support, we use a common logical hardware representation at network nodes. We use the Java Virtual Machine (JVM) for this purpose (see Fig. 3.2).

The major functions of the NRE includes the following: (i) to receive, instantiate and execute Netlet services as independent threads of control; and (ii) to identify incoming packets that require processing and assign them to appropriate Netlet services, which should process them.

3.1.2 Netlet Distribution

Two approaches to code distribution used by existing active networking systems are: (i) loading from the source [62]; or (ii) loading from a code server [69];

The "load from source" approach does not scale to a large user population. This is because each individual end application will have to download and host service codes locally before usage. This operation is performed even if the intermediate nodes host the required service, thus introducing redundant traffic within the network. This approach also has the disadvantage that the code must be kept in a persistent storage at the source, which is sometimes impractical (e.g. in a handheld device).

Alternatively, the use of code servers to host network services has been proposed. In this model, code servers are arranged in a hierarchy, similar to DNS nodes in the current Internet. When a network node requires a new service, it performs a search through this hierarchy to locate and download the desired module.

The major problems that arises when working with this approach are: (i) code servers will have to be configured to co-operate with neighbouring repositories in the hierarchy; (ii) the request does not always follow the shortest path route to the server hosting the required service; and (iii) those nodes that are higher in the hierarchy become points of implosion under heavy loading conditions. We believe that a decentralised solution is more appropriate, taking into account the ever expanding size of networks.

In the Netlet model, the server function of hosting Netlet services is distributed across:

- the intermediate network nodes; and
- the Netlet "home nodes" located at network edges.

By "home node" of a Netlet, we refer to a node that is responsible for per-

manently hosting the Netlet service. The key motivation for adopting a decentralised solution is that it eliminates the central points of failure associated with the centralised approaches. Furthermore, the use of home nodes, allows the service requests to be sent to a node that is sure to host the required service. Additionally, by exploiting the Netlet stores available at intermediate network nodes, the delay to obtain services can be minimised. We defer the actual procedure followed to discover Netlets in the Internet to section 3.3.2.

Netlet Labelling Scheme: A mechanism is also needed to identify individual Netlet services uniquely in a wide area network environment, such as the Internet. We propose to follow the URI naming scheme [119] for this purpose. In the Netlets architecture, names for individual Netlet services are derived by concatenating: (i) the address of the home node of that Netlet; and (ii) its service specific label name. These attributes are set by the home node and do not change for the lifetime of a Netlet. An example of a Netlet label is shown in Fig. 3.3. The home-node of this Netlet has the address "192.168.254.1" and its service specific label is "packet-forwarding-netlet". This Uniform Resource Identifier (URI) scheme coupled with network level routing support is used to support the discovery mechanism as in section 3.3.2.

Figure 3.3: Netlet Naming Scheme

3.2 Implementation of the Netlet Runtime Environment

In this section, we discuss the implementation details of the Netlet prototype model that we built using the Java language. The prototype supports two service

deployment schemes; they are: (i) **proactive deployment** – where Netlets migrate under their own control to deploy services at distributed points in the network; and (ii) **reactive deployment** – where services are deployed based on requests generated by the Netlet nodes. The Netlet prototype supports packet capture and processing (both header and content level processing). When appropriate, we use pseudo code representations to explain our implementation.

The Netlet Runtime Environment (NRE) consists of three major components (Fig. 3.4). They are: (i) Netlet Services; (ii) the Netlet Management Engine (NME); and (iii) the Packet Communication Engine (PCE). The basic architecture of an NRE is shown in Fig. 3.4.

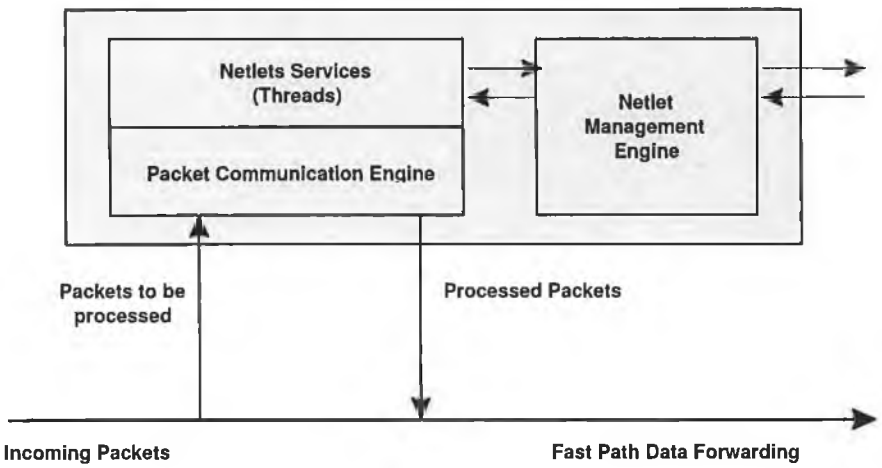


Figure 3.4: Architecture of a Netlet Runtime Environment

Netlet Services

A Netlet is a collection of Java classes that implements a predefined network service. Netlets operating in the network are both active and mobile. Netlet services are active in the sense that they are executed as independent packet processing threads within an NRE. Netlet mobility is achieved using the serialization facility available in Java.

Each Netlet service is derived from the base class, *Netlet*, and must implement

the $main()$ method, which is the entry point for a Netlet. The base Netlet class implements the $java.io.Serializable$ class. This allows state information of Netlet services to be captured and transported between network nodes. The methods that we developed to represent the base Netlet class are shown in Fig. 3.5.

Method		Description
void	Netlet (String netletName, Object load)	Create a Netlet
String	getName( )	Get NetletName
void	main ( )	Main method of a Netlet
void	setResume (String methodName)	Set the method to continue at new node
String	getResume ( )	Get the method to continue at new node
void	migrateTo (String dest, String resume)	Migrate to a given destination
boolean	isValid ( )	Retrurns validity of Netlet
void	setValidity (boolean cond)	Sets the validity of Netlet at the node
boolean	clone (String dest, String resume)	Clone the netlet
boolean	suspend ( )	Suspend the Netlet
Packet	readFrom (packetBuffer)	Read packet from the service queue
void	sendJarFile ( )	Send all classes required to launch Netlet
void	setCredentials ( )	Set the credential of the Netlet
void	getCredentials ( )	Get the credentials of the Netlet

Figure 3.5: API for building Netlets

Netlet Management Engine

The Netlet Management Engine (NME) must receive, instantiate and execute Netlet services as independent packet processing threads at Netlet nodes. The NME is also responsible for managing the Netlet services operating at a node. The management functions include: (i) applying security checks and granting appropriate permissions for incoming Netlets; (ii) monitoring and managing the resources among the Netlets resident at the node; and (iii) relocating/removing redundant services to ensure resource availability for future service requests. The key methods that we developed to implement the NME are listed in Fig. 3.6.

Method		Description
InetAddres	getInetAddress ( )	Get IP Address of Node
int	getNetletSendPort ()	Get port used to send Netlet
int	getNetletRecvPort ( )	Get port used to receive Netlet
void	pack (Netlet n, OutputStream outStream)	Pack a Netlt for sending
void	sendNetlet (Netlet, Address, Port)	Send Netlet to a given Address/Port
void	addHandler ( )	Add handler to process Netlets
Object	getHandler ( )	Handler to process incoming Netlets
Netlet	receiveNetlet (InputStream in)	Receive a Netlet
Netlet	unpack (InputStream inStream)	Unpack an incoming Netlet
void	register (netletName, resume, time)	Register a new Netlet
void	createPacketBuffer (netletName)	Create Packet Buffer
void	deletePacketBuffer (netletName)	Delete Packet Buffer
void	NetletThread (Netlet n, Resume method)	Start a Thread for a Netlet
boolean	activate (netletname,String resume)	Activate a dormant service
void	inValidate (netletName)	Delete a live service
byte[ ]	getServiceList ( )	Get currently available services
Interface	accessRights (netletName)	Set Access Rights
void	upLoadClasses(String NetletName)	Uploads all Netlet specific-service classes
void	downLoadClass(fromAddr, className)	Download a specific service class fromAddr
Object	getBuffer (NetletName)	Returns packet buffer corresponding to Netlet

Figure 3.6: API of the Netlet Management Engine

Packet Communication Engine

The role of the Packet Communication Engine (PCE) includes to: (i) identify and capture those packets that require processing at Netlet nodes; (ii) assign those packets to appropriate Netlet services for processing; and (iii) initiate the process of Netlet discovery, when a miss is recorded for a required service. We employed the Java based packet capture (Jpcap[1]) library to provide packet capturing facility at Netlet nodes. The Jpcap package provides Java based wrapper functions for the Berkeley Packet Filter (BPF) [120]. The API of the PCE is shown in Fig. 3.7.

[1]http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html

Method		Description
String	getDeviceList ( )	List of interfaces available for capturing
Jpcap	openDevice (String device, int promisc)	Initialises a network device for capture
void	setFilter (String filterCondition)	Filter for packet capture
void	capture (int count)	Start capturing packets
Packet	getPacket ( )	Returns a captured packet
void	handlePacket (Packet p)	Analyses a captured packet
void	IPAddress (int version, InetAddress dst)	Creates an IP packet
void	setIPv4Parameters(String [ ] flags)	Creates a packet with specified parameters
void	UDPPacket (String [ ] flags)	Creates a UDP Packet
void	TCPPacket (String [ ] flags)	Creates a TCP Packet
void	sendPacket (IPPacket packet)	Sends a packet over the network interface

Figure 3.7: API for Packet Capture

Starting the Netlet Runtime Environment

When the NRE is started it performs the following set of tasks, to ensure Netlet operation and packet processing support. First, it creates a portal object to receive and send Netlet services. We assume all nodes in the network employ a globally unique port number for this purpose. Next, it stores the information about all the available services at the node into its Netlets service structure. Finally, it instantiates the predefined filter rules that were registered by Netlets for receiving packets from specific traffic classes as presented in section 3.2.2.

3.2.1 Netlet Deployment

The above discussion presented the set of methods that were developed to support Netlet execution at network nodes. Now, we present in detail the procedure followed for deployment of Netlet services in the network. Recall that Netlets are either deployed based on the proactive deployment scheme or the reactive deployment scheme. Here, we describe the implementation level details of both approaches. Note, the current implementation of the Netlet prototype only provides weak mobility support.

```
public void migrateTo (String dst, String resumeMethod) {
    setResume(String resumeMethod);
    ME.sendNetlet (this, dstAddress,port);
}
```

Figure 3.8: Pseudo-code for Migration

The Proactive Service Deployment Model

In the proactive service deployment model, Netlet services autonomously migrate between network nodes to deploy networking services. For example, lets assume a Netlet, nLet, is operating at node N_A. Furthermore, we assume that the nLet service must migrate to N_B in order to accomplish a specified task (e.g. to install a new networking service). The steps followed during the process of migration are presented below.

Sending Node: The nLet service requiring to migrate from N_A to N_B, makes a call to its $migrateTo()$ method specifying the destination address as N_B and the name of the method to invoke at the other end (Fig. 3.8). The $migrateTo()$ method in turn calls the $sendNetlet()$ of the NME object (Fig. 3.9). This $sendNetlet()$ method embodies function calls to the $uploadClasses()$ and $pack()$ methods. The $uploadClasses()$ method transmits (i) the credentials of the service; and (ii) the complete set of classes required for the operation of the Netlet at the other end, as a Java archive file (jar[2]). Afterwards, the $pack()$ method serialises the state of the Netlet and sends it to the destination as an object stream. In order to ensure reliable communication, we employ TCP as the underlying transport level protocol for transferring Netlet services. The pseudo code for this process is presented in Figures 3.8 and 3.9.

Receiving Node: The NME on receiving the name and credentials of a Netlet

[2]This allows all the classes a Netlet needs to be transported in a single transaction.

```
public void sendNetlet (Netlet nLet, InetAddress dstAddr, int port){

    Socket outSock;
    try{
        outSock = = new Socket(dstAddr,port);
    }
    catch(UnknownHostException e){
        e.printStackTrace();
    }

    OutputStream outStream = client.getOutputStream();
    // send the name of the Netlet
    (new DataOutputStream(outSream)).writeUTF(nLet.getName());

    // send the credentials of the Netlet
    (new DataOutputStream(outSream)).(nLet.getCredentials());

    // send the jar file of the Netlet
    (new DataOutputStream(outSream)).(nLet.sendJarFiles());

    // use the Netlet Handler to send the object graph using java.io.serialisation
    NetletHandlerStore store = store.handlerFor(nLet.getName());
    netletHandler.pack(nLet,outStream);

    nLet.setValidity(false);
}
```

Figure 3.9: Pseudo-code for Sending a Netlet

checks its local security policies to determine whether the Netlet is permitted to be executed locally. If the Netlet qualifies[3], the NME accepts the classes of the incoming Netlet services.

Next, it creates a service object for the incoming Netlet. Following this, it makes a call to the $Netlethandler()$ method specifying the name of the service to be unpacked and initiated (Fig. 3.11). The $Netlethandler()$ makes an up call to the $unpack()$ method to rebuild the Netlet object from the incoming object stream (Fig. 3.12). On recreating the object, the $unpack()$ method using the Java Reflection API identifies the class name of the object and the main method, $main()$, that must be called to allow the object to resume execution. Based on this information, an independent thread of control for the Netlet service is launched (Fig. 3.13).

The Reactive Service Deployment Model

In the reactive service deployment model, requests for Netlets are generated by the Netlet nodes in the network. For example, we assume that a Netlet, nLet, is available at node N_A and that an incoming packet at node N_B requests processing using the nLet service. Subsequently, N_B initiates a service discovery process (described in section 3.3.2) to locate and download the nLet service. The service discovery process locates nLet at node N_A and requests it to send a copy of the service to N_B.

Next, N_A notifies the requesting node, N_B, that it will provide the nLet service. Furthermore, it sends the relevant Netlet to N_B. NME at node N_A packs and sends the service classes (i.e. the bytecode files of the Netlet service) as a jar file to node N_B. The $upLoadClasses()$ method in the NME API is used for this purpose. For this case there is no state information to be transferred.

The NME at N_B on receiving the notification message from N_A, sets up a custom class loader for downloading and constructing the Netlet from the incoming

[3]The implementation does not yet provide any security mechanisms. We used dummy strings to enact the role of Netlet credentials.

```
public class NetletReceive {

  public static int backlog;

  public ServerSocket rcvSocket;

  public NetletHandlerStore handlers;

  public NetletReceive( InetAddress addr, int port,  NetletHandlerStore store) {
      server = new ServerSocket ( port, backlog, addr) ;
      handlers = store ;
  }

  public Netlet receiveNetlet ( ) throws NetletException, IOException{

      Socket client = rcvSocket.accept ( ) ;
      InputStream inStream = client.getInputStream( );
      ...

       // Read the Netlet Name
      String  netletName = (new DataInputStream (inStream) ).readUTF( );

       // Get the credentials of the Netlet
       dos = new DataInputStream(new BufferedInputStream
         (new FileInputStream(credentialFile),128));

       // Receive the jar file containing the required Netlet classes
       rcvJarFiles( ) ;
       ...

       // call the Netlet handler to unpack the Netlet Object
      NetletHandlerStore store = store.handlerFor ( netletName) ;
      Netlet newNetlet    =    store.unpack ( inStream);

      // Make the incoming object valid in the service space
      netNetlet. setValid (true);
      inStream.close ( );
      return newNetlet
  }

}
```

Figure 3.10: Pseudo-code for Receiving a Netlet

```
import netlet.*;
import java.io.*;
import java.reflect.*;

public  NetletHandler extends SerializationHandler {
        public void unpack (InputStream) {
        }

        private  static class NetletThread extends Thread {
        }
}
```

Figure 3.11: Pseudo-code for Autonomous Operation of a Netlet

```
public void unpack (InputStream inStream) throws NetletException, IOException{

    MobileObject mObj = super.unpack(inStream);

    Netlet nLet ;

    try{
       nLet = (Netlet) mObj;
    }
    catch (ClassCastException e) {
    }

    try{
       String resumeMethod = nLet.getResume();
       Class cl = nLet.getPayLoad().getClass();
       Method method = cl.getDeclaredMethod(resumeMethod,null);
       new NetletThread(nLet,method);

    }

    catch(){

    }

}
```

Figure 3.12: Pseudo-code for Unpacking a Netlet

```
private static class NetletThread extends Thread{

    private Netlet nLet;
    private Method method;

    public NetletThread (Netlet n, Method m){
      nLet = n;
      method = m;
      start ();
    }

    public void run (){
      try{
        method.invoke(nLet.getLoad(),null);
      }

      catch (Exception e) {
          e.printStackTrace();
      }
    }
}
```

Figure 3.13: Pseudo-code for Starting a Netlet

data stream. The NME uses the $downloadNetletClasses()$ method for this purpose.

The NME then makes an up call to the $activate()$ method specifying the name of the service, the main class and the main method to instantiate the new Netlet service. This call results in the $activate()$ method creating a new Netlet object for the service. A call is made to the $NetletThread()$ specifying a reference to the object and the main method to instantiate the service. Following, this an independent thread of control for the Netlet is then launched (as per Fig. 3.13).

3.2.2 Packet Processing in the Netlets Network

In the previous section, we described the procedure followed to deploy Netlets in the network. Here, we explain the mechanisms followed to provide packet processing support using the deployed Netlets.

Packet Processing in the NRE

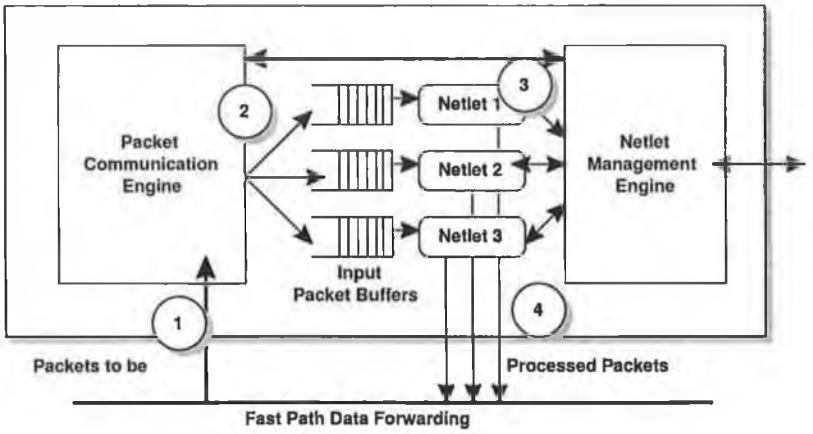


Figure 3.14: Packet Processing at the NRE

Packet processing in the NRE is either based on: (a) the reactive approach; or (b) the proactive approach.

- Reactive processing – In this model, packets include the name of the Netlet service that should process them at intermediate network nodes. Furthermore, by setting the router alert option field [121], such packets distinguish themselves from regular data packets that do not require processing within the network.

- Proactive processing – In this model, Netlet services instigate packet processing at intermediate network nodes, to process traffic flows belonging to users who deployed them. Here, Netlet services register packet filter rules

with the PCE, which specifies the flow details[4] that each Netlet requires to process (e.g. of this Netlet services are presented in Chapter 4).

The NRE provides packet processing support as follows. The PCE monitors for packets: (i) with the the router alert option field set; or (ii) which have a matching entry in the packet filter table of Netlet services. When a packet qualifies for processing, the PCE performs the following procedure:

- it captures the packet from the fast data path of the Netlet node (step as in as in Fig. 3.14);
- Reactive case: If the packet has its router alert option field set, the PCE then extracts the name of the service from the IP options field of the packet. Following this, it performs a call to the $getBuffer()$ object that contains the list of Netlet services that are available at the node. This object returns a reference to the input packet buffer object that corresponds to this specific Netlet. The PCE then writes the packet to this buffer object (step 2 in Fig. 3.14).
- Proactive case: If the packet only has a matching filter rule, the PCE performs a lookup in the Netlet filter table to identify the corresponding service that should process this packet. Following this, it finds and assigns the packet to the appropriate buffer object that corresponds to the service.
- Individual Netlet services loop forever receiving, processing and forwarding packets at Netlet nodes (step 3 and 4). Furthermore, they are in an idle state when no packets are available in their corresponding packet buffers. Note, individual Netlet services handle errors themselves that arise during packet processing.

[4]A flows specification can be constructed using (i) the pair of source and destination addresses; (ii) the pair of source and destination ports; and (iii) a protocol number.

Proactive Packet Processing

When an incoming packet has a matching entry in the filter table, the procedure followed by the PCE to process them is as presented in Fig. 3.15.

```
Hashtable filterSpecTable; // this table has mapping between packet filters and corresponding Netlets
Vector listAllServices; // this vector lists all Netlets at the node


String labelName;

if(labelName==null) {
    // Step-1: construct the packet filter identifier
    fSpec.setParameters(packet.getSrcAddress(),packet.getDstAddress(),
        packet.getSrcPort(),packet.getDstPort());

    // Step-2: check the filterSpec Table
    if(filterSpecTable.containsKey(fSpec)) {

        // Step-3: the service is active; write the packet to the buffer of the service
        labelName = filterSpecTable.get(fSpec);

        // Step-1: check whether the service is available
        Object nLetInfoObject = listAllServices.get(labelName);

        //Step-2: get status of the service active/dormant
        if((nLetInfoObject.getStatus())!=-1){
            // the status is active
            PacketBuffer packetBuffer = infoObject.getPacketBuffer();
            packetBuffer.writeTo(packet);
        }


        if((infoObject.getStatus())==-1){
            // the status is inactive
            packetBuffer = new PacketBuffer();
            packetBuffer.writeTo(packet);
            String resumeClassName = (listAllServices.get(index)).getMaInClassName();
            String resumeMethod = (listAllServices.get(index)).getResumeName();
            activate(labelName, resumeClassName, resumeMethod);

            // change the state of the object as active
            nLetInfoObject= (listAllServices.get(index)).setActive()
            listAllServices.add(index, nLet);
        }
    }

}

else{
        //Step-3: discard the packet
        discard(packet);
}
```

Figure 3.15: Pseudo-code for Proactive Packet Processing

Reactive Packet Processing

When in the case of an incoming packet has a reference to a Netlet service, the procedure followed by the PCE is as presented in Fig. 3.16.

3.3 Netlet Deployment

3.3.1 Proactive Service Deployment

In the proactive service deployment model, Netlet services are required to migrate under their own control and deploy new services at various points in the network in order to provide packet processing support on behalf of the end users who deployed them. For this purpose, Netlet services must discover Netlet nodes available at suitable locations (e.g. in various domains) to host the service. A typical example of such a service is the data caching service in IP multicast [47]. These services must be present at judicious points within the network so as to integrate reliable communication support for existing IP multicast protocols.

For example, in Fig. 3.17, the server node must install data caching service logic in three different domains, A, B and C. To deploy service in those domains, the server node must be aware of the location of the active nodes within each domain. However, due to the heterogeneous nature of the Internet (see Fig. 3.17-a), not all nodes in the Internet will be active. Thus both active and non-active nodes are expected to coexist in the future.

A possible solution is to use existing service discovery protocols such as Jini [122] or SLP [123]. In this approach, active nodes present in each domain can register their details (e.g. domain name, ip address, execution environment support available) with a central lookup server. Those nodes requiring to deploy services will be able to query and extract the list of active nodes that are present at various domains in the network.

However, the major problem with this approach is its lack of scalability when

```
// contains the list of Netlet services at a node,
      //corresponding state and reference to packet buffer object
Vector listAllServices;

// the packet contains a netlet name
if(labelName.toString() !=null){

      if (listAllServices.contains(labelName.toString())) {

            // Step-1: check whether the service is available
            Object nLetInfoObject = listAllServices.get(labelName);

            //Step-2: get status of the service active/dormant
            if((nLetInfoObject.getStatus())!=-1){
                  // the status is active
                  PacketBuffer packetBuffer = infoObject.getPacketBuffer();
                  packetBuffer.writeTo(packet);
            }


            if((infoObject.getStatus())==-1){
                  // the status is inactive
                  packetBuffer = new PacketBuffer();
                  packetBuffer.writeTo(packet);
                  String resumeClassName = (listAllServices.get(index)).getMainClassName();
                  String resumeMethod = (listAllServices.get(index)).getResumeName();
                  activate(labelName, resumeClassName, resumeMethod);

                  // change the state of the object as active
                  nLetInfoObject= (listAllServices.get(index)).setActive()
                  listAllServices.add(index, nLet);
            }
      }


      // simply forward the service perform service discovery
      else{

            // forward the packet
            socketOut.send(packet);

            // perform service discovery
            Object nLetInfoObject = new infoObject(labelName,packetBuffer)
            listAllServices.add(nLetInfoObject);
            netletDiscovery (labelName);
      }

}
```

Figure 3.16: Pseudo-code for Reactive Packet Processing

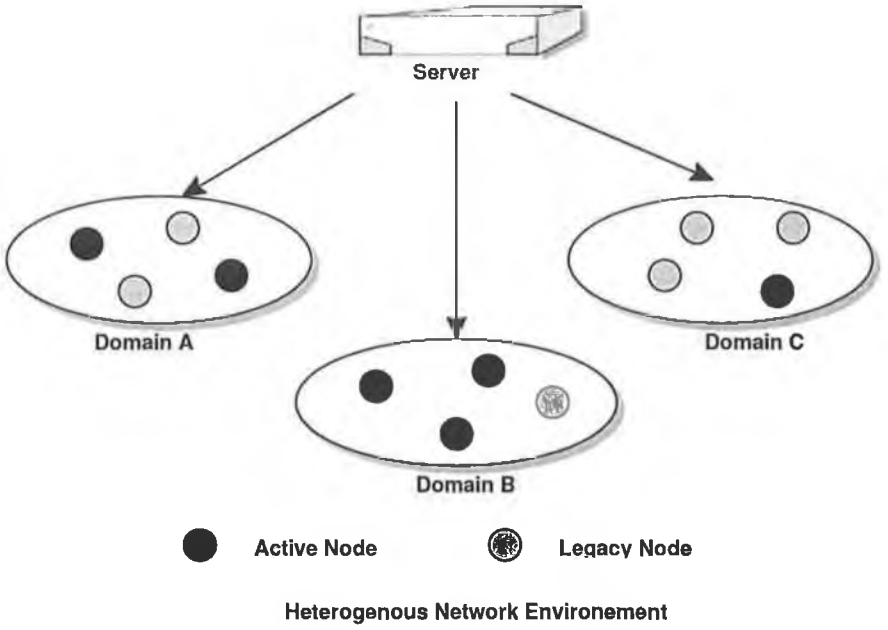


Figure 3.17: Proactive Service Deployment

working with wide area networks such as the Internet. Also, this system is not fault tolerant. Thus, a scalable discovery scheme is required. Below I propose a DNS based scheme to locate active nodes in the Internet.

DNS-based Discovery Scheme

Domain Name System (DNS) servers can be used to obtain a list of existing hosts located in a domain. This feature can be exploited to discover active nodes present in a network domain. The set of active network nodes that are present under a common administrative control can be listed in the node reachability information list of their corresponding domain servers.

By using existing DNS query tools[5] such as nslookup or dig, the host list of a domain can be retrieved. The information retrieved includes both the host names and the corresponding addresses of the nodes. By making the host names self-descriptive with standard prefix formats, the list of active nodes present in a net-

[5]http://www.dns.net/dnsrd/tools.html

work domain can be extracted. For example, a name such as *active.netlet-node-32.dcu.ie* can be used to represent an active node of type Netlet present in the *dcu.ie* domain.

If, by co-incidence, a passive node matches the prefix format, this scheme will include it in the list of active nodes. Valid active nodes can be identified by exchanging *hello* messages between the server and the nodes on the list.

Algorithm for Discovery of Active Nodes on an End-to-end Path

In addition to locating active nodes on a per-domain basis, it may also be necessary to locate active nodes on an end-to-end path basis for service deployment purposes. Fig. 3.18 shows the algorithm to support discovery of active nodes on an end-to-end path. The key idea here is to find the end-to-end path between the server and the client (say using a tool such as traceroute) and then to recursively query each domain on the end-to-end path to identify active nodes within them.

```
traceroute to destination (obtain the domain names of nodes on the end-to-end path)
  for all (domains on the path [ ]) {
     active_node_address [ ][ ]  =  dns_tool ( domain_name [ ])
  }
```

Figure 3.18: Algorithm for Active Node Discovery and Service Deployment

Note, performing discovery of active nodes on a per-user basis will not scale to a large user population. However, the above algorithm is only intended to work with applications which deploy services that manage groups of users rather than individual end clients. In Chapter 5, I describe applications in the areas of multicast and server selection that require such support in the Internet.

Benefits of the DNS based Approach

– this approach leverages an existing Internet protocol;

- this approach is completely distributed and does not suffer scalability concerns; and

- it is reliable and fault tolerant;

Overall, the DNS-based approach provides a scalable means to discover active node support in the Internet.

3.3.2 Reactive Service Deployment

In the reactive model, data packets identify the Netlet service, which should process them at intermediate network nodes by name. When a Netlet node is presented with a request for a service, the node will be able to either provide the service: (i) immediately – if the requested service is available locally; or (ii) after an initial delay – caused due to discovery, deployment and instantiation of the service locally. Setting up services on-the-fly involves the following costs:

(a) increased memory requirements – the packets that arrive during service discovery phase will have to be buffered until the service is instantiated locally; and

(b) increased initial delay experienced by those packets that arrive when the service is not available locally;

An analytical model is presented in Appendix A.1 to study the dynamics of the reactive service deployment scheme as observed by end user applications. In Fig.3.19, the procedure followed to setup a service at a Netlet node is presented. On arrival of packets that require active processing (step1 in Fig.3.19), the node checks to see whether the required service is available locally. If the requested service is present (i.e. a hit), packets are queued for processing. If a miss is recorded, a discovery scheme to locate the service is invoked (step 2). Packets that arrive during the process of service discovery are queued in the dormant packet buffers

(step 3) for later processing. Note a timeout function is associated with the dormant packet buffers. If the required service is not discovered within the specified timeout value, packets in the buffer are dropped.

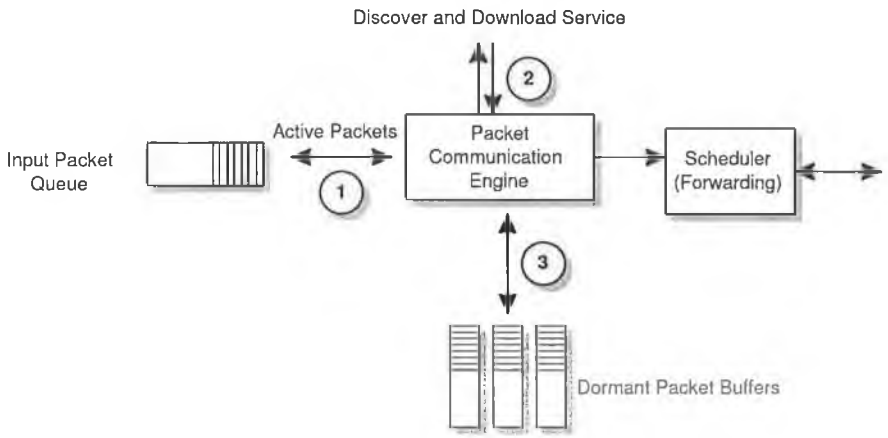


Figure 3.19: Packet Processing at a Netlet Node

Thus, the delay to dynamically discover and deploy Netlets can affect the overall quality of service perceived by end applications. An efficient protocol for service discovery is thus required. Below a new protocol is proposed which performs better than the "load from source" and code-server approaches (discussed in section 3.1.2).

Stigmergy: A Scalable Protocol for Wide Area Service Discovery

The key feature of the Stigmergy protocol is that each Autonomous System in the network is treated as an independent two level caching structure in which the upper level, $L1$, contains pointers to Netlet services that are present in the lower level, $L0$. For example in Fig. 3.20, information about the Netlet services that are present at nodes 1 to 5, are stored at level $L1$, which can subsequently be used by other nodes in the network for service discovery.

However, many issues have to be addressed to realise such a system. They are:

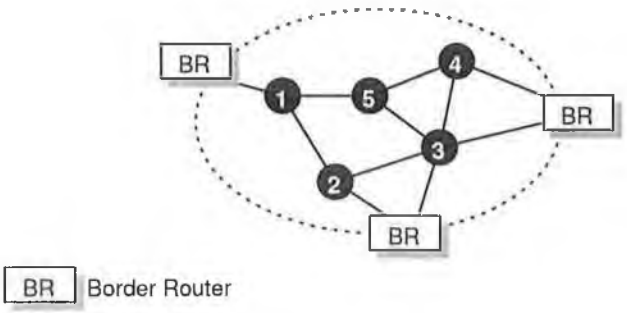


Figure 3.20: Domain Level $L1$ Cache

(a) Which node(s) in an Autonomous System should be elected to function as $L1$ point(s) ?

(b) How to construct the $L1$ level representation for an Autonomous System?

(c) What would be the criteria used to decide whether the node which requires a service should initiate a discovery process, (i.e. contact distributed $L1$ levels in the network) or contact the home node of the Netlet directly?

(d) What search strategy should be adopted to route service requests through various $L1$ levels within the network?

(a) Border Routing Nodes as Aggregation Points: The border router nodes of an Autonomous System are the most suitable points to act as $L1$ locations for the following reasons. A routing path for packet in the Internet comprises segments that span different Autonomous Systems. Individual Autonomous Systems contain both interior and exterior routing nodes (border nodes). The former route packets within the domain, while the latter perform inter-domain routing, i.e. between neighbouring Autonomous System (Fig. 3.20). Thus, when a packet needs to traverse an Autonomous System, it must be routed through one of its border nodes. Thus, by storing pointers at border routing nodes of an Autonomous System, we can create an exact map of the services available within it.

(b) Self-organising Network Caches: The list of Netlet services present at caches of each individual Netlet node can be announced to the border router nodes of

the domain[6]. The border routing nodes can record these announcements into the "$L1$ Table". The information recorded into the $L1$ Table should contain the name of the Netlet service and the address of the node on which it currently resides. Thus, the model follows a self-organising technique to build the information for the $L1$ level.

(c & d) The Combined Model: The decision as to whether a node requiring a service must search the network caches or contact the home node directly is not clear cut. One approach is described below.

The node which requires the service sends a request to the home node. Should the request pass through a border node which contains a reference to the service, the border node suppresses the request and deals with it itself. Otherwise the request will reach the home node for processing.

Service Discovery support using Network Routing

The Stigmergy protocol for performing service discovery is presented below. A Netlet node (node S in Fig. 3.21) on facing a miss for a Netlet generates a request to the home node (node H) of the required Netlet service (say X). Recall that the address of the home node for a Netlet can be extracted from its name (section 3.1.2). The format of the request packet is shown in Fig. 3.22.

The request packet is routed along the shortest path route to the home node. In Fig. 3.21, the shortest path route from S to H is, S-a-B1-B2-b-c-d-B3-H. When the packet reaches an border router[7] (e.g. either B1 or B2), it performs the following operations.

First, it extracts the name of required Netlet service from the packet. Next, it checks whether there is a corresponding entry for the service in its $L1$ Table. If the result is true (i.e. a hit), the border router signals the node holding the service

[6]It is assumed that each node in a domain is aware of the address list of its border routers

[7]Its assumed that Stigmergy packets are recognised by border routing nodes based on a unique protocol number, while intermediate network nodes route them as regular data packets.

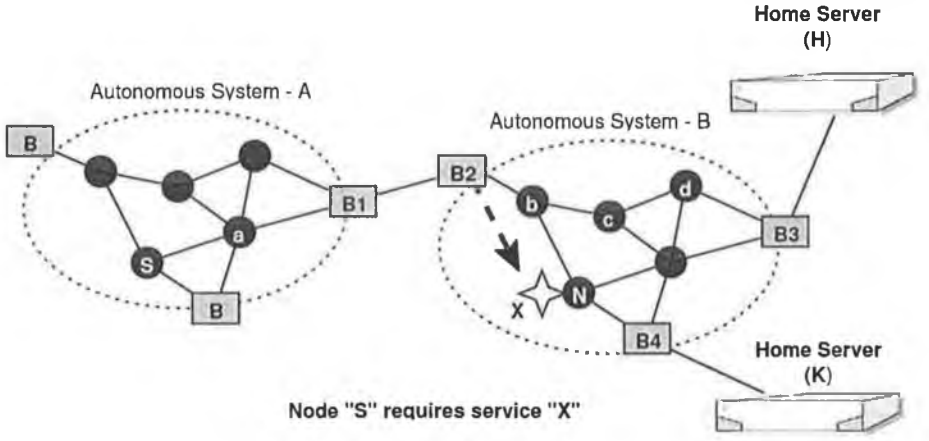


Figure 3.21: An Example of Service Discovery in ABC Framework

Protocol Number (Service Request)	Source Address (Requesting Node)	Destination Address (Home Server for the Netlet)	Netlet Name

Figure 3.22: Format of a Request Packet in Stigmergy

to send a copy of the service to the node requiring the service code (i.e. to the source address of the packet). However, when there is no entry in the $L1$ Table (i.e. a miss), the border node simply forwards the packet on the shortest routing path towards the packet's destination address, i.e. the home node of the Netlet.

In the case of B1 as in Fig. 3.21, it does not contain an entry for the service. Hence, it simply forwards the packet to H. However, when the packet reaches B2, it identifies that the required service is present at node N. Hence, B2 signals N to deliver the requested service to node S. Thus, the border node functions as a packet deflector service within the network for service discovery packets.

Discussion

Minimising State at Border Routers: In the above approach, each border router of an Autonomous System stores pointers for all the services present within it. This state information at border routers can be reduced by exploiting the phenomenon of route aggregation present in the Internet [124]. For example consid-

ering the Autonomous System B in Fig. 3.21, the shortest route paths from N to H and K are through $B3$ and $B4$ respectively. In this case, routing requests for H, would always traverse $B3$. Based on this fact, $B3$ can restrict itself to caching only those entries of services which belong to H, while $B4$ can restrict itself to entries of K. Recall that names of Netlet services include the address of the home node. This feature can be exploited to achieve an optimal storage model for the Stigmergy protocol.

Signalling Service Announcements: When multiple border routers exists in an Autonomous System, Netlet nodes are required to send service announcements (i.e. about new services and evicting old services) to all of them. Some of the possible mechanisms that can be employed are: (i) simplify to initiate unicast connections between the Netlet node and each border router; or (ii) the border routers can be grouped under a local multicast group, to which Netlet nodes can announce; or (iii) a service announcing Netlet can be launched to visit the border routers sequentially and inform them of the new service.

Status of Routing Nodes: Routers may go out of service abruptly without prior announcements due to node or link failures. Hence, border routers should avoid assigning discovery requests to those nodes that have announced participation in the $L1$ cache, but are later unavailable for service. For this purpose, the $L1$ Table must also record the status of the Netlet nodes that are participating. The periodic routing updates sent by Interior Gateway Protocols (e.g. RIP, OSPF) can be used for this purpose.

Multiple Instances of a single Netlet: When border routers contain multiple entries that corresponds to the same service in its $L1$ Table, they can assign the incoming service discovery requests in a round robin or a random manner. This would allow load balancing to be performed among the nodes in the network.

Benefits of the Stigmergy Protocol

Large Virtual Caches: This protocol, by self-organising network nodes that are under a common administrative control into virtual cache clusters, maximises the chances of discovering the required service locally.

Zero Cache Cooperation: The Stigmergy protocol is completely distributed and follows a best-effort cache co-operation model. By this we mean that Netlet nodes can join/leave a virtual cache group depending on a best-effort basis. Furthermore, this architecture avoids the need to configure and maintain independent caching framework for service discovery purposes.

3.4 Summary

Netlets are autonomous, nomadic mobile components which persist and roam in the network independently, providing predefined network services. Netlet nodes offer runtime environments for the operation of Netlet services. I first presented in this chapter a flavour of the Netlets architecture and its working. Next, I presented the set of methods that I developed to support the execution of Netlet services at network nodes. I then described the service deployment mechanisms supported by the prototype. Measurements of the prototype performance will be presented in Chapter 5.

I also proposed a DNS-based discovery scheme to locate active nodes in the Internet. This approach leveraged an existing protocol, namely DNS . The scheme features a distributed architecture and does not suffer scalability concerns. Finally, I proposed a service discovery protocol, referred to as Stigmergy, which supports the discovery of Netlet services in the network. The Stigmergy protocol is completely distributed and follows a best-effort cache co-operation model. Furthermore, this protocol avoids the need to configure and maintain independent caching frameworks for service discovery purposes. Experiments to assess

the performance of these service deployment schemes will be presented in Chapter 5.

Chapter 4

Netlets for Multimedia Applications

High-end multimedia applications such as digital television, scientific visualisation, medical imaging and advanced collaborative environments impose more stringent requirements on the support mechanisms provided by the underlying networks than still media such as text, images and graphics. In these applications the quality and the timeliness of the content being delivered is crucial. Furthermore, such applications require globally distributed user groups to be interconnected in a scalable manner.

The unicast-based best effort service model of the current Internet is not adequate for supporting distributed multimedia applications that involves multiparty communication. In this chapter I present solutions to a variety of problems that occur when supporting multimedia applications in the Internet. The areas considered are: (i) Quality of Service (QoS); (ii) Multicast; and (iii) Server Selection. These solutions are intended to assist a graceful migration from the best-effort model to an Internet technology with multimedia support.

4.1 RSVP Reservation Gaps

4.1.1 The Problem

High-end networking applications such as e-commerce, multimedia, distributed data analysis and advanced collaborative environments feature demanding end-to-end quality of service (QoS) requirements. QoS refers to the capability of a network to provide priority processing including dedicated bandwidth, controlled jitter and latency, and improved loss characteristics to selected traffic classes (audio, video, etc.). Some of the major approaches proposed to retrofit the Internet with QoS capabilities include Integrated services/RSVP (InServ/RSVP) [2, 125, 126], Differentiated services (DiffServ) [127], MultiProtocol Label Switching (MPLS) [128], Traffic Engineering (TE) [129], and Constraint-based routing [130]. These enhancements to the Internet are intended to provide end-to-end QoS support.

The Internet is a heterogeneous network environment interconnecting different autonomous network systems on a global scale. Due to this, the future availability of QoS support features at all nodes in the Internet is highly unlikely. Hence, non-QoS nodes will coexist with QoS-supporting nodes in the network. Throughout the discussion I will refer to the latter as "Q-nodes" and the path segments which interconnect them as Q-segments. The flows requiring QoS guarantees are referred to as "Q-flows". If a mixture of both Q and non-Q segments is present along a Q-flow's path, no global end-to-end service levels can be guaranteed. I refer to the non-Q segments present along a Q-flow's path as *Reservation Gaps*. Mechanisms to compensate for the possible impact on QoS of these reservation gaps will be required if QoS-sensitive applications are to be deployed as widely as possible in the Internet.

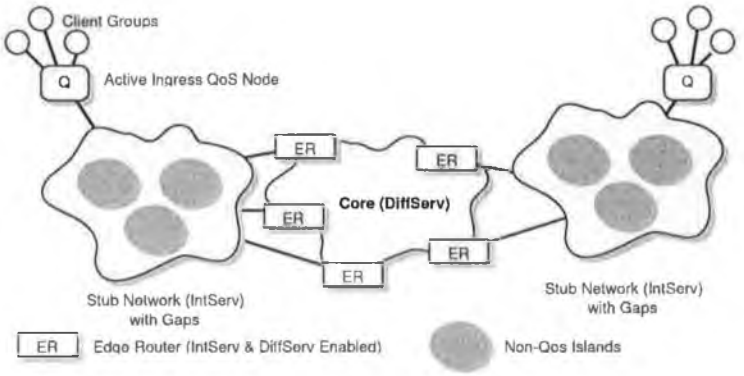


Figure 4.1: IntServ Over DiffServ QoS Model With Non-QoS Islands

4.1.2 RSVP Reservation Gaps

For the purpose of illustration, I describe reservation gaps in the context of the RSVP protocol [2]. The IntServ over DiffServ framework [131] provides a scalable end-to-end QoS model for the Internet. In this architecture, the stub network domains are based on an IntServ network model while the core network follows a DiffServ [127] based architecture. This approach is currently one of the most valuable solutions for end-to-end QoS provisioning, since it combines the benefits of both the IntServ [2] and the DiffServ [127] architectures. This model provides QoS signalling capabilities for resource reservation by end-user applications and also provides good scalability properties when working in the core network. The reference architecture (see Fig. 4.1) which I have used to describe the approach for enabling robust QoS support in the Internet is based on this end-to-end QoS model.

Due to the heterogeneity exhibited by the Internet, a route from source to destination for a Q-flow may not be available which is comprised exclusively of QoS supporting path segments. Hence the flow must traverse one or more reservation gaps. QoS signalling protocols like RSVP [2] for IntServ networks provide support for operation in heterogenous networks. When non Q-nodes (i.e. non-RSVP nodes) are present along a Q-flow's path, the RSVP signalling

messages reserve required resources at the RSVP nodes (Q-nodes) and rely on the available best-effort service offered across the non Q-nodes. Here the focus is on the reservation gaps caused by the non-QoS islands in the stub network domains (see Fig. 4.1).

In an environment with reservation gaps, it would be possible to provide end-to-end QoS support either by:

- restricting Q-flows to paths comprising exclusively Q-nodes – with a low population of Q-nodes, this approach will fail; or
- permitting Q-flows to traverse reservation gaps while assuming that best-effort service provides adequate QoS across non-Q segments – this approach will fail during congestion periods.

A more realistic solution would be to minimise the number of reservation gaps present along a Q-flows path whilst monitoring the unavoidable gaps so as to provide information about the availability of resources across the gaps.

A solution to overcome deficient reservations was presented in [132] using a receiver-initiated agent-based approach. In that approach, individual receiver end applications initiated mobile agents for solving the problem of deficient reservations, when reservation gaps (referred to as reservation tunnels in [132]) were detected along the end-to-end path of a Q-flow. The tunnel detection mechanism used was to back-trace and probe every node on the session's end-to-end path to identify the existence of tunnel segments. On identifying the exact location of the tunnel(s), agents were deployed by the end application to monitor the tunnel characteristics and to report the measurements to the application. In comparison, the approach presented here is scalable and is distinguished by its capacity to continuously monitor and adapt to network conditions.

4.1.3 The Role of Netlet Nodes

The solution presented here requires additional features (to support management and monitoring of gaps) to be present at the Q-nodes in the network, specifically at those Q-nodes that sandwich gaps. I use Netlets for this purpose. The reasons for using Netlets are: (i) a Q-node will not act as an entry/exit node for a reservation gap on a semi-permanent basis, but rather on a dynamic basis, as dictated by the changing network state and the operation of its routing protocols. Thus it is advantageous to implement the necessary functions using dynamically loadable modules; (ii) The start and termination point of reservation gaps can only occur at the boundaries between QoS and best-effort regions of the network; as QoS support is rolled out, the locations of these boundaries will change as will the Q-nodes which must support reservation gaps. Manual deployment of the necessary software to handle the dynamically formed reservation gaps creates a management problem which is avoided in the Netlets architecture.

In the discussions below, I assume that Q-nodes (supporting RSVP) in the Internet are also able to support Netlet services. This is a reasonable assumption, since the technology to support QoS in networks is evolving and the software (e.g. QoS signalling) on such nodes is likely to be subject to regular upgrades, as part of which Netlet support may be added. To support monitoring of non Q-segments, I assume all nodes in the network support SNMP [79].

4.1.4 Application Level Support

For the sake of generality, I make the approach independent of the end application's in-built QoS features. Thus a general assumption I make is that ingress nodes connecting the users to the stub network are QoS provisioned. Note an approach to provide QoS support to non-QoS aware applications using Netlets is presented later in this thesis.

4.1.5 Robust Reservation Support

This approach to providing robust support for reservations in the presence of reservation gaps is based on the following three mechanisms: (a) discovering the reservation gaps; (b) monitoring each gap; (c) managing the Q-flows traversing the gap.

Fig. 4.2 depicts reservation gaps caused by a non-QoS island, G, in an IntServ network. For the example, let's consider a sender node (S1) generating RSVP PATH messages destined to a receiver node. In this case, nodes Q1 and Q3 that sandwich the reservation gap Gap1 (caused by non-QoS island G) are referred to as the entry and exit nodes of the gap. In the case of a completely non-QoS provisioned stub network, the maximum path length of a reservation-gap will span from the Ingress/Gateway QoS Node (e.g. node Q) to an Edge Node (e.g. ER) of the DiffServ domain, which is always both RSVP and DiffServ enabled.

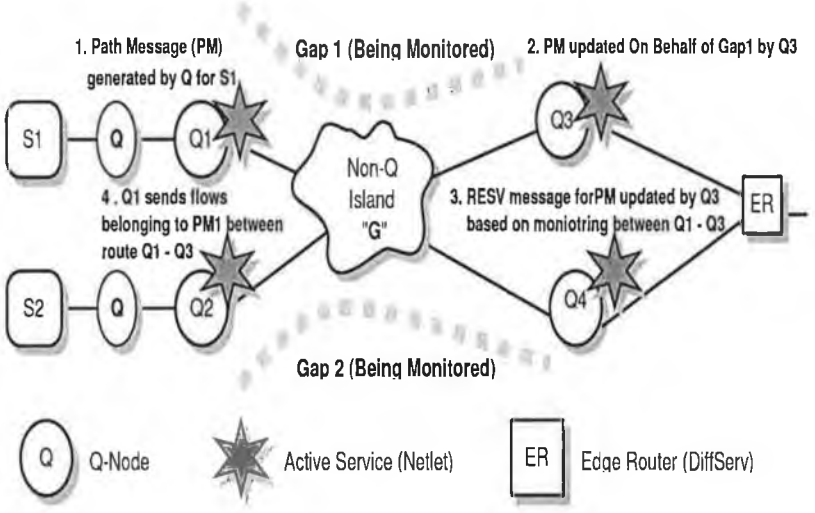


Figure 4.2: The Reservation Gap in an IntServ Network

4.1.5.1 Dynamically Discovering Reservation-Gaps

The PATH messages of the RSVP protocol are used to discover the reservation gaps present along a Q-flow's path. Each PATH message in the RSVP protocol

includes the address of the last known RSVP-capable node in the Phop (previous hop) field. If the node receiving the PATH message does not have direct connectivity to the Phop node (based on the information available in the neighbourhood table[1]), it recognises the existence of a reservation gap between itself and the node identified as the Phop node. For example in Fig 4.2, when a downstream node such as Q3 receives a PATH message from an upstream sender node, S1, node Q1 is identified as the Phop node, thus identifying the presence of a gap.

4.1.5.2 Monitoring the Reservation-Gap

On discovering the existence of a reservation gap, the relevant exit node (e.g. Q3 for Gap1 in Fig. 4.2), takes on the role of managing the gap. First, it fetches the required Netlet code using the service discovery protocol (section 3.3.2) and installs the service. Following this, this Netlet service discovers the entry Q-node of the gap and clones and autonomously installs itself at the entry node. Those Netlet services present at the entry and exit Q-nodes of the gap are referred to as *Entry-active-service* and *Exit-active-service* respectively. These services co-operate to perform monitoring of the reservation gap (for example, the services installed at Q1 & Q3 co-operate to monitor Gap1, in Fig. 4.2).

SNMP [79] is used to measure the path characteristics of the gap. The *Entry-active-service*, generates SNMP messages with destination *Exit-active-service*. The SNMP messages traverse the reservation gap collecting relevant metrics (for example, the available bandwidth, queue length, discard rates, interface utilisation etc.) by probing the relevant MIB entries of the non Q-nodes. The delay in traversing the gap is measured by sending time stamped packets from the *Entry-active-service* to the *Exit-active-service*. The direction of travel of the SNMP messages and delay measurement packets conforms with the travel direction of the PATH message along the reservation-gap (for example from Q1 to Q3 for a PATH message

[1]A table listing the nodes to which a host node is directly connected.

from S1 in Fig.4.2). Continuous packet processing will be necessary, for as long as there are non-Q-flows present in the reservation-gap.

4.1.5.3 Managing the Q-Flows Traversing the Gap

Managing the Q-flows involves intercepting and updating the RSVP signalling messages (PATH and RESV) traversing the reservation gaps to reflect the available resources in the gap. The PATH message primarily functions to install reverse routing state in each router along the path, and secondly to provide receivers with information about the characteristics of the sender traffic and end-to-end path so as that they can make appropriate reservations. The RESV messages in turn carry reservation requests to reserve resources based on the PATH message content. In the conventional reservation scheme, the PATH and RESV message are ignored along the uncontrolled reservation-gaps. In contrast, by monitoring the gap we can provide the receiver and sender nodes with accurate information about the path characteristics and reservation availability.

Q-flow management across the reservation gap is accomplished as follows. The end host operating over the end-to-end QoS model (e.g. node S1 or Q in Fig. 4.2) sends PATH messages destined to the receiver node with a request for a reservation (Step 1 in Fig. 4.2). The Q-nodes (RSVP) along the path create path-state information at the local node for each PATH message and update the AD-Spec object (this advertises the path characteristics to the receiver). When a PATH message is received at an exit node (Q3) of a reservation-gap, this node updates the PATH message with information gathered using the monitoring scheme (Step 2 in Fig. 4.2). Hence, the receiver node receives accurate state information regarding the end-to-end path. When the exit node receives the request for resource reservation (the RESV message) it checks for the availability of resources (e.g. bandwidth) and conformance to the delay constraints specified. On acceptance of the reservation by the exit node, this node informs the entry Q-node to send all

data-flows belonging to the request across the exit node until the PATH state information of the flow present at the entry Q-node times out (Step 4). This is done to mimic the RSVP behaviour followed in maintaining soft-state routes along the end-to-end path. This will allow the data packets of a flow to follow the same path across the gap as that of the delay measurement packets and the RSVP signalling messages. The entry node sends data flows to the prescribed exit node by setting the Source Route Option present in the IP header of packets belonging to the flow. For example, Q-flows originating from S1 and whose signalling messages traverse Gap1 have the Source Route Option set to Q3. When the data packets reach the exit node, the Source Route Option is cleared and the packets are forwarded. This allows multiple gaps present along the path of a Q-flow to use the same Source Route Option field.

4.1.6 Routing Algorithms To Minimise The Number of Gaps Along A Q-Flow's Path

The Netlet based approach as described above can provide end-to-end QoS support model in a heterogeneous network environment with reservation gaps. In today's Internet RIP [133] and OSPF [134] are the two most widely used Interior Gateway Protocols (IGP). Both compute the shortest path (SP) between source and destination. When working in a heterogenous environment, such as the Internet, the shortest path may consist of an arbitrary number of both "Q" and non-Q nodes. However, it will be more efficient if the routing mechanism select paths for Q-flows with the minimal number of reservation gaps. I[2] propose two route selection algorithms that aim to select paths with the maximum number of Q-nodes:

most reliable – shortest path algorithm (MR-S) – this selects a set of minimum hop count paths and, where there is more than one such path, selects the

[2]jointly with my colleague Karol Kowalik

one with the maximum number of Q-nodes. If there are several such *most reliable – shortest* paths, random selection is used.

shortest – most reliable path algorithm (S-MR) – this selects a set of paths with the maximum number of Q-nodes and, where there is more than one such path, selects the one with the minimum hop count. If there are several such *shortest – most reliable* paths, random selection is used.

I assume here that non-Q nodes forward packets according to the decisions of existing routing protocols (such as RIP or OSPF) while the Q-nodes use the MR-S or S-MR routing algorithms.

4.1.7 Remarks

The unpredictable behaviour of traffic within the non-QoS path segments present along a Q-flow's path and the inability to support reservations across them can cause problems in providing end-to-end service guarantees in the Internet. I have described a Netlets based approach to build a robust end-to-end QoS support model. I also proposed routing enhancements (*MR-S* and *S-MR*) that when employed at Q-nodes select a path for the Q-flow with the minimum number of reservation gaps. This technique features excellent dynamics and scales for large networks and user populations. The good dynamics make support of short lived Q-flows feasible. The control traffic generated (to monitor and manage the non-QoS path segments) is confined to the corresponding reservation gap, thus reducing congestion and packet loss. Overall, this solution provides a mechanism to support robust end-to-end QoS support in heterogeneous network environments such as the Internet.

4.2 Transparent and Scalable Client-side Server Selection using Netlets

4.2.1 The Problem

With the user population of the Internet continuously on the rise, the demand for web based services is also witnessing a corresponding exponential rise in demand. Content replication at multiple locations in the network has been identified as a scalable means to provide clients with improved service response time, reliability and performance levels. When replicated servers are available at multiple locations in the network, clients are presented with the problem of dynamically choosing the best or the optimum performing server for service provisioning. Most server selection methods [9–14] proposed to date work to distribute load across servers.

Techniques that perform load distribution across servers were traditionally designed for load balancing across server clusters. In such approaches, the selection decision is purely based on the server load. However, when working to assign client requests to distributed server systems, the location of servers with respect to client nodes has been found to affect the service response time perceived by clients [135, 136]. This is due to the characteristics of the network path segments through which requests get routed. Hence, making server selection decisions based on the client's view of the network and server conditions is appropriate.

To facilitate this, web servers hosting replicas at multiple locations in the Internet provide users with the address list of servers available for service provisioning. Current approaches followed by clients for selecting servers from a replicated set include: (a) random selection; (b) directing requests always to a fixed server; or (c) to choose the closest server according to geographical proximity. However, the above mentioned approaches have proved to offer poor server

selection solutions [135–137]. In some cases, clients also try parallel downloads of the same document from multiple servers. In this approach, once a server accomplishes the requested task, the other requests are terminated. This approach generates redundant network traffic thus consuming excess bandwidth.

Research efforts have been made to identify client-side server selection metrics that support efficient server selection in the Internet [135–137]. It is proposed that the clients themselves would perform the requisite measurements and make the selection decisions. There are two major shortcoming of such techniques: (i) such solutions are not scalable because every client on network will use measurement probes that will consume network resources; and (ii) the servers are unable to influence selection decisions, so that it is not possible to support request distribution across the available servers.

In [138], a modified web browser referred to as the smart client, was used to perform server selection. This client software downloads an applet supplied by the service provider to realise service-specific routing. This approach creates increased network traffic due to applet downloads and the corresponding communications which ensue between the applet and the servers.

4.2.2 Goals

A solution based on Netlets will now be presented. This solution is intended to meet the following goals:

- **Load Distribution:** it provides a mechanism to distribute client requests for content among multiple, possibly geographically dispersed, servers;
- **Client-side based Service Decision:** assigning requests to a specific server occurs close to a client, to maximise service responsiveness;
- **Server Customised Selection Techniques:** the selection of a server is based on metrics supplied by the servers, allowing eg., load balancing or link

bandwidth probing to be performed;

- **Scalability:** avoiding the use of measurement probes generated by individual clients, and employing aggregated set of measurements that can be used for client communities;

- **Demand-based Service Support:** providing selection services at those locations where potential client communities for the service exist;

- **Service Location Transparency:** clients request content from a single address, so that the operation of scheme is completely transparent to the client; and

- **Fault Transparency:** the solution is robust, with no single point of failure.

4.2.3 Solution Overview

The reference architecture that is employed to demonstrate the solution is shown in Fig. 4.3. Here, a heterogeneous network environment in which both active and legacy routing nodes exist is assumed. Netlet based services embedded with intelligence to support server selection are deployed by servers close to potential client communities to setup dynamic service decision points within the network. I refer to those network services that support server selection as the *director services*. Each service decision point transparently directs client requests to the best performing server based on its in-built intelligence supported by real-time measurements that are performed between itself and server replicas. I propose to deploy director services based on user demand. The exact location and the number of director services present in a network is dictated by the location of the relevant communities of interest in the network (see section 4.2.6).

4.2.4 Dynamic Setup Of Virtual Primary Server

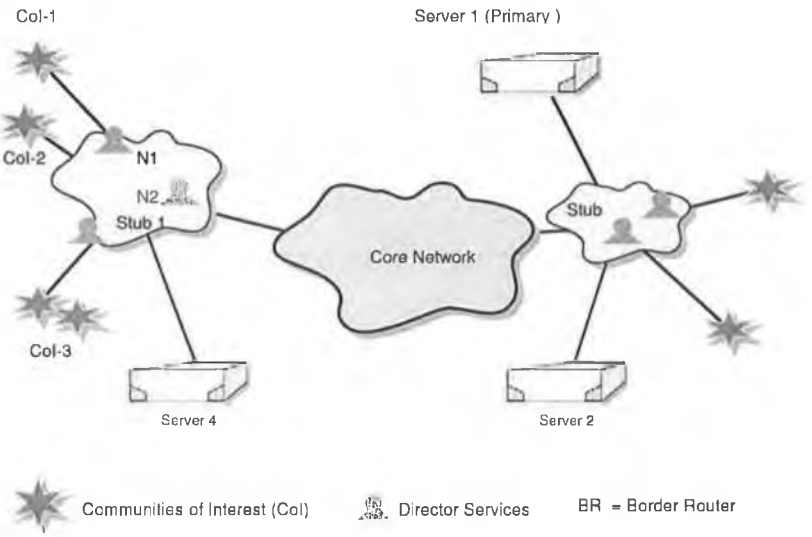


Figure 4.3: Replicated Servers and Communities of Interest

Anycast based Director Services Anycasting is defined in [11] as: "a service which provides a stateless best effort delivery of an anycast datagram to at least one host, and preferably only one host, which serves the anycast address". IP anycast [11] is a network service that allows a client to connect to the nearest of the receivers that share the same anycast address. "Nearest" is defined in terms of network distance metrics.

In the Netlets based approach to server selection, an anycast address is shared among the Netlet based director services (i.e. inherently the Netlet node at which the service operates) that act as service decision points and with the primary content server. Clients are only aware of the director service location rather than the individual server replicas. Consequently, client requests that correspond to anycast addresses are automatically routed to the closest service decision point rather than directly to a server.

The representation of address sharing in the Netlet based scheme is shown in Fig. 4.4. The director services share an anycast address, while the replicated servers have distinct IP addresses. The primary server shares the anycast address with director services and also has a distinct IP address. This feature of binding

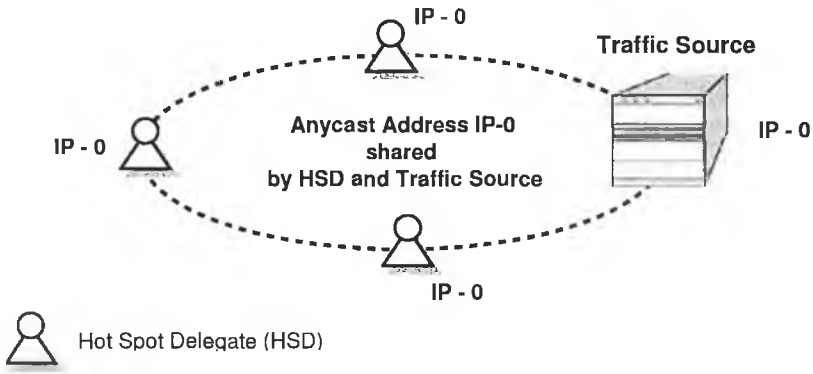


Figure 4.4: Address Sharing in the Netlet Scheme

two addresses to the primary server, allows a client request to get automatically routed to a server if no service decision points exist close to its location.

Service Deployment of Director Services The mobile and autonomous property of service code in the Netlets architecture avoids manual intervention for service deployment. To introduce service selection support at multiple points in the network, the director service is informed with the address list of nodes requiring service. This Netlet then autonomously migrates to each node and installs service thus avoiding centralised deployment schemes and generating less network traffic. Methods to find exact locations to providing director service support and the scheme to discover active nodes at those locations are presented in section.4.2.6.

Registration of Director Services at Netlet Nodes: When a director service is deployed at a Netlet node, this service requests the local node: (i) to register for receiving client requests that correspond to the anycast address for which the Netlet holds the permission; and (ii) to advertise routes for the anycast address.

The concept of virtual host and interfaces used by IP aliasing can be used to register director services at a Netlet node. IP Aliasing is simply a mechanism that enables a single physical or virtual network port to assume responsibility for more than one IP address. For example, in a linux based router, a simple com-

mand such as, *ifconfig eth0:<virtual interface number > <anycast ip> <netmask>* can be used for this purpose. By using this feature a Netlet node will be able to support multiple director services simultaneously. New routes to anycast addresses can be advertised as part of normal routing table updates.

4.2.5 Server Selection using Director Services

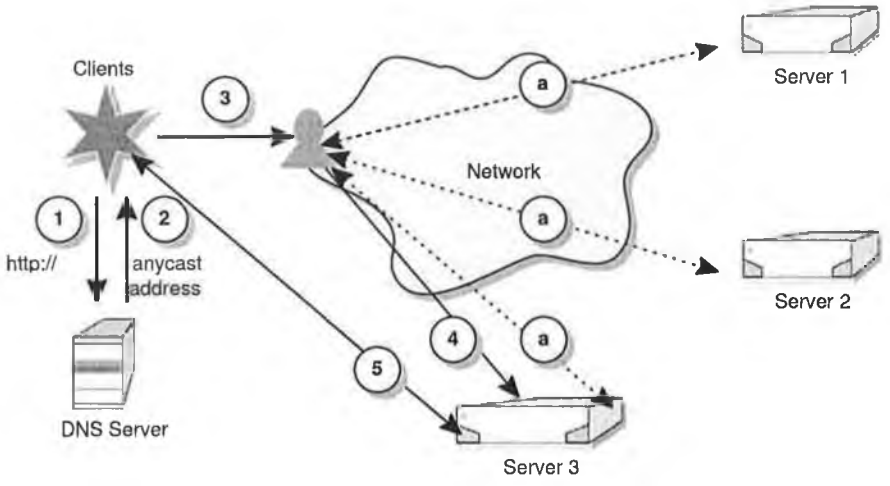


Figure 4.5: Transparent Server Selection using Director Services

Here, I describe the mechanism used to transparently direct client requests to the optimal server. For the example below, it's assumed that TCP is used as the transport protocol.

When a client wants to connect to a server, the client performs a name resolution query to the DNS server (step 1). The reply from the DNS node consists of an anycast address (step 2) which refers to the distributed server group. The client sends a TCP SYN packet to this address to initiate a connection. This packet is automatically routed to the closest service decision point (i.e. director service) that corresponds to the server group (step3 in Fig. 4.5). On receiving the request, the director service selects the optimum server (step 4) based on selection metrics (step a). The selection metrics are described in section.5.2. Hence, the request is directed to the chosen server (for example server 3 as in Fig. 4.5).

Note that the SYN packet from the client has the anycast address as its destination address. Hence, a mechanism is required to direct the SYN packet transparently to the chosen server. One solution to this problem is to encapsulate the SYN packet within a unicast packet destined to the unicast address of the selected server. Unique protocol identifiers can be used to identify such encapsulated packets at the server end. The server on receiving the SYN packet replies with the SYN-ACK packet directly to the client based on the available destination address.

Stateful connections (step 5) can then be maintained with the selected server using route pinning. In this approach, the server receiving the anycast packet pins the route (using IP Source Route Option field) for future packets originating from the client during that session to pass through the unicast address of the selected server. With modifications performed at the TCP/IP control blocks at the server side, when such packets are received, the IP block passes it to the request processing daemon. Stateful connections may be alternatively maintained over UDP.

4.2.6 Supporting Architectural Features

In this section the architectural features required to support the Netlets based approach to server selection are discussed. This feature set includes: (i) a method to support discovery of locations requiring director service support; (ii) discovery of active nodes at those locations; and (iii) scalable routing for anycast addresses using unicast routing protocols.

Communities of Interest: A deployment scheme is required for distributing director services within the network. Analysis of access logs of various web servers have shown the existence of *communities of interest* in the Internet [139–142]. These are groups of clients which are responsible for generating a high proportion of the

workload on servers and which are geographically close or under common administrative control. Servers should deploy director services close to such communities.

In [139], a network-aware method based on prefixes and netmask information gathered from Border Gateway Protocol (BGP) routing table snapshots was used to identify client clusters (referred as communities of interest here) in the Internet. The authors validated the BGP based technique to locate communities of interest by employing two approaches based on "domain name" and "traceroute". This technique gave good performance even when used with historical snapshot data.

The results from [139] based on globally collected web server logs show that 90% of communities have 100% of their clients topologically close to each other. It was also reported that around 5% of communities accounted for the majority of the clients and for generating a high percentage of the workload on the web server (see Fig. 4.6). This confirms earlier studies [141] that claim the existence of Zipf-like distributions in a variety of web measurements.

By being able to locate communities of interest, servers will be able to provide transparent selection support to the majority of the client population that use the services. Remaining clients are served directly by the primary server. Since there are relatively few clients outside the communities of interest, this does not represent a major burden on the primary server. We adapt the above described BGP based technique to locate communities of interest present in the network to support director service deployment.

Hot Spot Nodes: The DNS-based scheme proposed in section 3.3.1 can be used to locate active network nodes present in the Internet. A suitable location for director service operation is at the ingress/internal routing nodes of the stub network (such as N1 and N2 in Fig. 4.3) through which users connect to the Internet. This is a consequence of the feature of route aggregation present in the Internet [124].

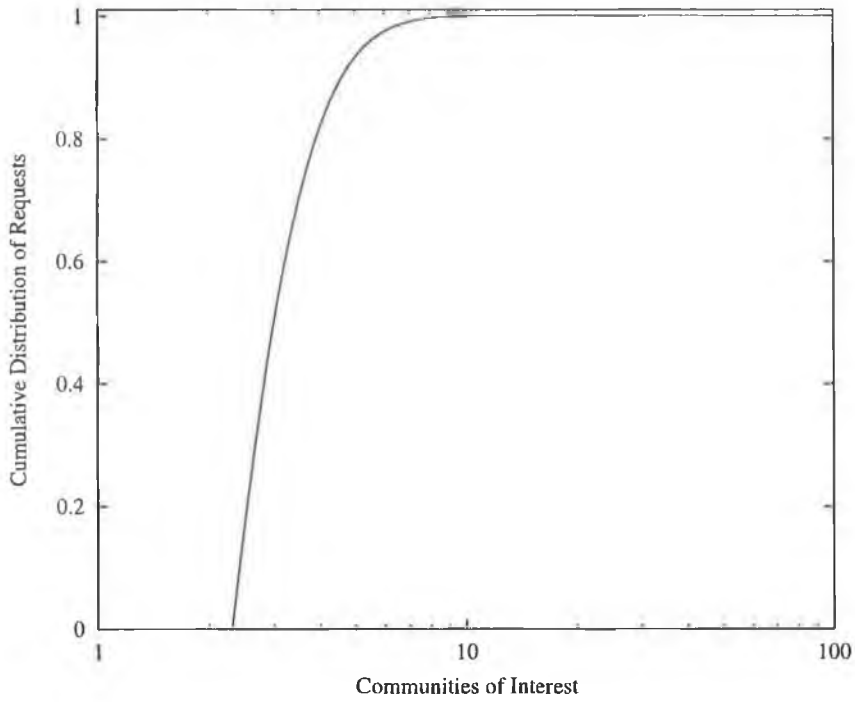


Figure 4.6: Cumulative Request Distribution Across Communities of Interest

For example in Fig. 4.3, let the director service, N2 direct requests to server S1 (IP addresses 192.15.36.12) and S3 (136.10.1.2) based on some predefined selection metric. Suppose the border router BR1 aggregate routing entries for destination IP addresses starting with 192 while BR2 serves for IP addresses with 136 as the start. In this example, the route over which N2 communicates with the servers will share many links with the corresponding routes for clients in the community CoI_1 which accesses the Internet via $stubnetwork1$. I refer to those Netlet nodes that act as judicious points for deployment of director services as "hot spot nodes" and their addresses as "hot spot addresses"(e.g. N1 is the hot spot location for users from CoI_1 while N2 is for clients present in communities CoI_2 and CoI_3 in Fig. 4.3).

The algorithm for locating and deploying director services in the network is presented in Fig. 4.7. The BGP based scheme [139] automatically generates the list of hot spot addresses in the Internet. Using the DNS-based approach 3.3.1,

```
hot_spot_domains = find Communities of Interest using BGP Clustering Method
    for all (hot _spot_domains) {
            active_node_address [ ] =  dns_tool ( hot_spot_domain [ ] )
              if (no active node @ hot_spot_domain)
                 {
                     Find Next Hop Domain That Connects User to Internet (traceroute )
                 }
             active_node_address [ ] =  dns_tool ( hot_spot_domain [ ] )
    }
    server_select_netlet_service . moveTo(active_nodes_addresses [ ] )
```

Figure 4.7: Algorithm for Active Node Discovery and Service Deployment

we will be able to discover corresponding hot spot nodes and their addresses.

If the domain that holds the client group fails to contain active nodes, the next hop domain within the stub network connecting the clients to the Internet is queried. Locating the name of the second domain can be performed using traditional network tools such as traceroute.

Scalability of Unicast Routing Protocol for Anycast Addresses: Netlet based director services employ global anycast addresses to seamlessly integrate the dynamically constructed service decision points with the client-server based web communication model. When director services are deployed within stub networks they behave as local anycast groups to the corresponding stub domain. Due to this specific nature of the Netlets approach, conventional intra-domain routing protocols will be sufficient to route packets destined to anycast receivers local to the domain. For example, distance-vector algorithms, such as RIP inherently provide support for anycast service [12].

Employing unicast protocols for anycast services causes each service decision point present within a stub network to take up an entry in the internal routing table. However, this approach is scalable because: (i) the number of service decision points within a network is driven by user demand local to that domain; and

(ii) routing nodes present in stub networks has more free memory resources and CPU cycles when compared to routing nodes present in core networks.

Recall that when no service decision points exist within a domain, the anycast packets are routed to the primary content server which shares the same anycast address with director services (Fig. 4.4). The inter-domain routing can be implemented in a scalable manner using the method of Global IP Anycast (GIA) [143]. GIA uses the notion of popular and unpopular anycast groups in the Internet. The popular groups refers to those sets of anycast addresses that are often accessed by users from a particular domain. However, for unpopular groups (here, those groups which are routed to the primary server), packets are routed to a default unicast address that is encoded within the 32-bit anycast address.

4.2.7 Benefits of Employing Director Service Netlets

Temporal Shifts in User Demand Across Communities of Interest: Analysis of commercial web server logs [142] have proved the existence of demand shifting across communities of interest in the Internet. The authors of this paper propose to allocate distributed resources on demand near client locations to support such variations. Complementarily, using the Netlets approach, director services will be able move in accordance with demand to support server selection. The intelligence to support such feature can be embedded in the director services themselves.

Scalability and Knowledge Sharing: Selection techniques based on using measurement probes by each client for server selection will not scale for large networks such as the Internet. The Netlet scheme offers to implement scalable server-customised probing techniques. For example, director services that belong to the same server group and operating in close vicinity (eg., the same stub domain) will be able to share measurement probes. Furthermore, director services can be scoped to probe only a reduced set of servers when the replica set comprises a

large server group.

Support for Wireless Network nodes: Wireless network nodes have constraints on the availability of local resources and power. Hence, supporting server selection software at such nodes will be inefficient. Furthermore, wireless nodes will be unable to participate in continuous communication with server groups to perform selection decisions. The Netlets scheme readily offers support for wireless nodes by implementing the decision procedure in the network rather than on the client nodes.

4.2.8 Remarks

I proposed a novel technique to support transparent and flexible server selection in the Internet. The Netlets based approach provides a client-side server selection solution which is server-customisable, scalable and fault transparent. This approach combines the benefits of anycast addressing with a mechanism allowing the adoption of any server selection algorithm. By using Netlets, service decision points can be deployed dynamically to the locations in the network where they can most efficiently serve a large number of clients. This approach makes the solution inherently scalable, since it minimises the amount of overhead generated by measurement probes.

4.3 MENU: Multicast Emulation using Netlets and Unicast

4.3.1 The Problem

Multimedia applications such as Internet TV and advanced collaborative environments have generated a demand for services that allow multiparty communication in the Internet. This will allow such applications to support data dis-

semination to large groups of users in a scalable and reliable manner. In contrast, the current Internet is predominantly based on the unicast based point-to-point communication model.

In 1990's Deering [3] proposed the multicast service model to support multi-party communication in the Internet. IP Multicast [3] is a network level service in which routers disseminate multiple copies of datagrams to interested group members. This approach to logically group dispersed receivers offers operational advantages for content and network providers by minimising network resource demands and end-system overheads. Despite extensive research [15–18], multicast routing protocols have not been widely deployed in the Internet.

One of the primary reasons that discourages widespread multicast deployment in the Internet is the lack of a scalable protocol model. Existing IP multicast protocols require routers in the core of the network to store per-flow state information and to support per-flow packet forwarding operations. Performing per-flow operations inside the core of the network affects the network scalability. This is because, as the number of simultaneously operating multicast sessions increases, there is a linear increase in state information which leads to increased packet processing delays and memory requirements. Given the amount of data flowing through the core of the network, any protocol which requires the maintenance of considerable state information is likely to prove impractical.

Some of the other major factors that discourage multicast deployment are the lack of : (i) reliable inter-domain multicast routing protocols; (ii) reliable communication support - the existing multicast model supports best-effort service and hence does not support reliable communication which limits the applicability of multicast in the Internet; and (iii) access control - conventional IP multicast protocols allow any node in the network to send/receive data to the group, facilitating flooding attacks. A detailed discussion of these problems can be found in [144].

Recent research efforts [145–147] have demonstrated a one-to-many abstrac-

tion of the basic multicast model that scales better than conventional IP multicast. Such a model is appropriate for large scale applications such as Internet TV, automatic software distribution, etc. The available single source multicast models e.g., [145, 146] have been successful in supporting a secured group communication model and in overcoming the Class D address depletion problem. However, they still lack scalability and reliability.

A solution using unicast to build multicast services was presented in [148]. The hard-wired nature of this approach means that the protocol model is non-extensible. Furthermore, each node on the multicast tree is required to maintain state information, which affects scalability. In [149], a combination of ephemeral states and unicast forwarding was employed to build multicast services. In this approach, receivers use a topology-probing mechanism to identify a graft point on the delivery tree and instantiate an active service at that point to duplicate and distribute incoming datagrams. This approach generates additional traffic and state information at intermediate network nodes to support continuous tree optimisation.

4.3.2 Goals of the MENU Protocol

Below, I present the MENU protocol which is intended to serve large scale single source multicast applications. MENU builds multicast support in the network using Netlets and unicast addresses. The key goals of the MENU model are:

- **To minimise processing requirements and the amount of state information in networks:** the protocol model should work with core network domains which do not store state. Furthermore, forwarding states should only be maintained at branch points of the traffic delivery tree;

- **To achieve scalability through route aggregation:** must support address aggregation with hierarchical address allocation to achieve scalability as in the unicast model;

- **Avoid Inter-Domain Multicast Routing Protocol:** should avoid the need for inter-domain multicast routing protocols for session establishment;

- **Avoidance of address collision:** overcome the need for global address allocation for each application, which otherwise will lead to address collision problems;

- **Incremental deployment:** allow gradual and transparent deployment of the protocol in the network without penalising or disrupting existing network services;

- **Reliability:** support recovery of lost data with minimal network overhead and receiver burden;

- **Secured group communication:** the sender node in a group must be authenticated, which will allow the model to be free from flooding attacks similar to Denial of Service attacks in IP networks;

- **Accommodating receiver heterogeneity:** support working with heterogeneous receivers and different service level requisitions; and

- **Fault transparency:** the solution must be robust, with no single point of failure.

4.3.3 MENU Protocol Concept

The reference architecture shown in Fig. 4.8-a captures the basic characteristics of the MENU protocol. The multicast delivery tree in MENU is a two level hierarchical structure where users are partitioned into client communities based on geographical proximity (e.g. CoI-1, CoI-2 in Fig.4.8-a). Each client community in the network is treated as a single virtual destination for traffic from the server. Netlet based services referred to as Hot Spot Delegates (HSDs), are deployed by servers at "hot spots" close to each client community to function as virtual traffic

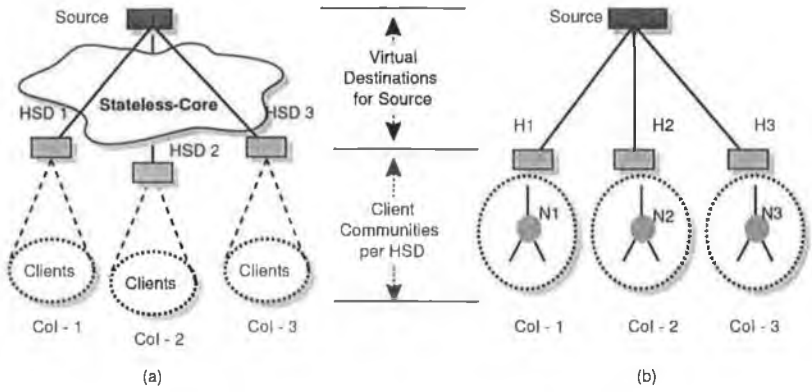


Figure 4.8: Two Level Hierarchical Model

destinations for the traffic from the server and also to act as virtual source nodes for all users in the community. The source node feeds data to these distributed HSDs which in turn forward data to all downstream users.

The primary reasons for pushing the tree building complexity to edge networks are as follows. Firstly, the scalability of any protocol which stores state in the core of the network may be poor. Secondly, research findings [148, 150] have shown that close to 70% of nodes in multicast trees have an average fan-out degree of 2 and only function as traffic relay nodes. These relay nodes are likely to be located near the source. Therefore there is little demand for packet replication in the core of the network. Finally, since routing nodes present at the network edges are likely to process fewer data flows at lower bit rates than core networks, the expense involved in supporting multicast there will be less.

4.3.4 Hot Spot Delegates

Hot Spot Nodes: A suitable location for HSD operation is at the ingress/internal routing nodes of the stub network through which users connect to the Internet. This is a consequence of the feature of route aggregation present in the Internet [124]. For the example in Fig. 4.8-b, H1 is the hot spot location for users from Col-1 while H2 is for clients present in community Col-2. The node on

which the HSD service operates is referred to as the Hot Spot Node (HSN). The exact location and the number of "hot spots" present in a network is dictated by the location of the relevant communities of interest in the network, as discussed in section 4.2.6. The DNS-based approach 3.3.1 can be used to discover Netlet nodes that will host the HSD services.

Deployment of HSD: A single HSD Netlet service is deployed in the network with the address list of nodes requiring service activation. This Netlet then autonomously migrates to each node and installs the service thus avoiding centralised deployment schemes and generating less network traffic.

Hot Spot Delegates as Virtual Destinations: The server maintains an address list of HSDs operating in the network. This address information includes the unicast address of the HSNs and the port on which the HSD receives data from the traffic server. Note that HSDs are not specific to any multicast session. When a server is required to support simultaneous multicast sessions in a network, it informs the HSD of the specific session details.

4.3.5 Hot Spot Delegates as Virtual Sources

Connection Attraction using an Anycast Address: MENU employs a global anycast address [143] to seamlessly integrate HSDs into the traditional client-server paradigm followed in the Internet. The MENU protocol shares this anycast address among the Hot Spot Delegates that act as virtual service points and with the traffic source node. Thus the distributed HSDs and the server are presented to the rest of the network as a single logical entity. Consequently, messages from users that correspond to a server's anycast address are automatically routed to the closest HSD rather than directly to the sender node. If no HSD exists close to a client's location, the messages get automatically routed to the source node itself.

Activation of the HSD at a Hot Spot Node (HSN): When a HSD service is deployed at a HSN, this service requests the local node: (i) to register for receiving client requests that correspond to the anycast address for which the HSD holds the permission; and (ii) to advertise routes for the anycast address. The concept of virtual host and interfaces used by IP aliasing can be used to register HSD services at the HSN. Mechanisms to register multiple addresses to a single network interface can be found in section 4.2.4.

4.3.6 MENU Protocol Details

The MENU protocol works as follows. A multicast session in MENU is identified by a globally unique anycast address (that corresponds to the traffic source e.g. a video server) and a source generated port number.

$$multicast\ session =< anycastaddress, portnumber >$$

When a user wishes to receive data from a server, the user connects to the server (using the available global anycast address) as in the unicast communication model. User requests that correspond to the server's anycast address are automatically routed to the closest HSD available rather than directly to the sender node. HSD services that receive join requests from users, in turn, generate HSD subscription messages to the source for receiving session data. The traffic source feeds data to these active HSDs in the network, which in turn forward data to its downstream user nodes (see Fig. 4.8-b).

Recursive Packet Replication and Forwarding using RepN: MENU performs recursive packet replication and forwarding within the network to distribute datagrams to members of a multicast session. To support recursive operation, multicast packets carry unicast destination addresses of immediate downstream branch nodes rather than Class D addresses as in the conventional IP multicast model.

Replication Netlet services, referred to as RepN, are employed for this purpose. These RepN service operating at branch points of the tree replicate incoming data packets to each of its downstream receiver member. Each replicated packet is set with the unicast destination addresses of the downstream receiver member. Furthermore, the RepN service sets the source address of the packet to the global anycast address and places the local node's unicast address into the IP Source Route Option field of the packet. The inclusion of two source addresses allows the downstream receivers: (i) to know the global session to which the packets belong; and (ii) to know the local upstream source responsible for packet replication. Packets are recursively replicated until they reach the end user nodes. For the example in Fig. 4.8-b, the source node serves HSDs H1, H2 & H3. Each HSD then transmits the datagram to its downstream members (N1, N2, & N3) with the source route option set to itself. Next, intermediate network nodes generate packet replicates with the source option set to N1, N2 & N3. Note that, if the node is a legacy routing node, it just forwards the packet towards the unicast destination address based on its routing table entries. This feature allows incremental deployment of the MENU protocol.

Replication Netlet (RepN): In addition to packet forwarding and replication support for multicast packets, the other functions of RepN services include:

- recording the list of live multicast sessions that traverse the local node in the Multicast Session Table (MST); the session details include the *global source address of the session* (i.e. the anycast address), *global port number*, *actual source address* (present in IP Source Route field) and *destination address of the packet*;
- evaluating whether the local node acts acts as a transit or a branch node in the tree;
- maintaining the list of downstream receiver node addresses in the receiver table (RT) for which the local node is the upstream source; and

- listening for join requests that corresponds to any live session present in the MST.

RepN Deployment using a Reactive Strategy: Multicast packets in MENU carry the name of the Netlet service that will process them at intermediate network nodes, i.e. RepN in this case. On arrival of a multicast packet, if the RepN service is not present locally, the active node triggers a request for service download. This request is sent to the actual source which replicated the multicast packet. Recall that the actual source address of a MENU packet is recorded in the IP Source Route Option field. This reactive strategy allows incremental deployment of RepN services within the network. The service activation delay is minimal. This is due to the fact that the node responsible for replicating the multicast packet is present locally.

4.3.7 Traffic Distribution from Server to HSDs

In the absence of state storage in the core network, traffic distribution from the server to the HSDs is handled using unicast connections only. However, if some routers in the core of the network are configured to handle multicast state information, the source can build a minimal state multicast tree as described in the next section. For the case below, it's assumed that the core network is stateless and only describe the procedure to build multicast tree from each HSD to the local users. However, when working with stateful core networks the same procedure can be used to build the multicast traffic delivery tree from the server to the HSDs.

4.3.8 A Reactive Approach for Delivery Tree Construction

Each HSD on receiving session data from the server, replicates and forwards datagrams to all downstream receivers that generated join requests for the ses-

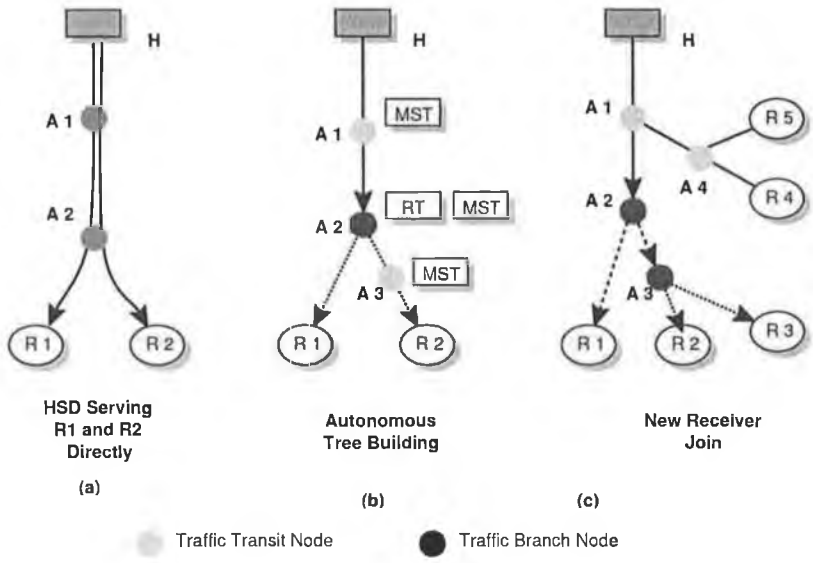


Figure 4.9: Traffic Distribution From HSD to Users

sion. (e.g. R1 and R2 in Fig. 4.9-a). The data packets as they travel towards the destination trigger the building of the traffic delivery tree from the HSD to all downstream receivers.

On-Tree Node in MENU: RepN services operating on the data flow path of the multicast packets record the session details in their MSTs. For example in Fig. 4.9-a, when individual data packets are sent from H to R1 and R2, RepN services at A1 & A2 record session details in their corresponding MSTs. Hence, routers on the delivery path automatically become members of the MENU tree.

Autonomous Tree Building: When an RepN service at an on-tree node records multiple flows that belong to the same session in its MST (e.g. in Fig. 4.9-b, where both A1 and A2 have two copies of the same flow from H), it initiates a simple procedure to evaluate whether the local node is either a transit or a branch point in the MENU tree. The evaluation procedure is as follows. The RepN service using the local routing table entries checks whether all such flows have a common next hop node;

- if true, *the RepN service knows that it is a transit node for the traffic, and does not*

perform any further evaluation;

- if false, *the RepN service knows that it has to act as a branch in the MENU tree and works as described below.*

Branch Node: The RepN service creates a Receiver Table (RT) and adds the addresses of all downstream receivers for which it is the optimal branch point. It then requests the actual source node (as recorded in the MST) currently serving these receivers to handover the session. Following this request, the actual source node adds this requesting node as a downstream member in its RT. Furthermore, the source node hand overs the set of receivers it was handling to this new optimal branch node. For the example in Fig. 4.9-b, the RepN at A2 on identifying itself as a branch node works as follows. This RepN service by consulting the MST identifies H as the actual source for the session. Next, A2 requests branch node status from H. Additionally, it advises H that it is the optimal branch point for downstream receivers R1 and R2. Next, H adds A2 as its immediate downstream member in its RT and performs handover of R1 and R2 to A2.

Joining a session: When a join request from a user to a live multicast session traverses an on-tree node, the RepN service captures the request and then works as follows: (i) if the node is a branch point in the tree, it adds the user as a downstream member in its RT; or (ii) if the node is currently acting as a transit point for traffic, the RepN service recognises that it has to act as an optimal branch point from now on. It then follows the procedure described above to claim branch node status. For example in Fig. 4.9-b, A3 is a transit node for the request from R3. After receiving the request, A3 in turn requests for a change in status from transit to branch from the actual source, A2. Furthermore, it specifies to A2 that it is an optimal branch for traffic to the downstream member R2.

Leaving a session: In MENU, it is assumed that periodic heartbeat messages are issued by users to their corresponding actual source nodes in order to receive session data continuously. Thus, when a user node wants to leave the session, it simply stops sending these heartbeat messages. After an appropriate timeout interval, the source removes the user from its RT. A branch node in MENU changes to a transit node only when the number of downstream receivers for that particular node falls below two. For the example in Fig. 4.9-c, when R2 leaves, A3 has only a single receiver R3. Following this event, the RepN service at A3 hands over R3, to the upstream source from which it has been receiving data for the session i.e. A2.

4.3.9 Dynamics of the MENU Protocol

Sub-optimal Branches for Transient Periods: At times, sub-optimal branches may exist due to race conditions in user joins. For example in Fig. 4.9-c, when R4 and R5 issue join requests in immediate successions, the branch node at A1 recognises itself as the upstream source for those receivers. However, the optimal branch point is A4. This sub-optimal structure arises because A4 has not been added as a member in the MENU tree. A4 will become a member in the tree only when multicast datagrams traverse it. Hence, during this intermediate transient periods sub-optimal branches may exist. However, once multicast data packets flow across A4, this node joins the tree automatically. It next recognises itself as a branch point and performs tree optimisation as described in section 4.3.8. Note that during this transient period, receivers may receive redundant data. By employing sequence numbers within MENU packets, receivers will be able to ignore such redundant packets.

Session Integrity and Avoidance of ACK/NACK Implosions: Receivers must send ACKS/NACKS for requesting retransmission of lost data to the actual source

that replicated the packet. Recall that in MENU multicast datagrams carry the address of the node which generated the packet in the Source Route Option field. This facilitates receiver nodes in sending ACK/NACK messages to the actual node which generated that packet. By being able to source route packets, modifications to protocol stacks at end-user nodes are not required, thus achieving service transparency. For example in Fig. 4.9-c, packets to R1 from *A2* are labelled as being from A2 in the IP Source Route Option and not from H. If the HSD itself is the source node of the packet (e.g. as H is the source for A2 in Fig. 4.9-c), it inserts the unicast address of the local node to ensure session integrity.

Reliable Communication using Data Caches within the Network: By employing the above described technique, Netlet nodes will not suffer ACK/NACK implosions. Furthermore, by providing network caches at intermediate network nodes recovery of lost data can be supported with minimal delay. The HSNs and the active routers can support processing and buffering resources to store session data for this purpose. For example, the model presented in [47] for retrofitting existing IP multicast protocols with reliability support can be used for this purpose.

4.3.10 Remarks

I proposed a new multicast protocol referred to as MENU. MENU builds a scalable multicast protocol model by pushing the tree building complexity to the edge network, thereby eliminating processing and state storage in the core of the network. MENU also provides reliable multicast communication services by supporting data caching within the network. Another architectural feature of MENU is that it automatically supports heterogeneous receivers; the Netlet nodes can perform media thinning within the network to suit end user terminals.

4.4 Transparent QoS signalling support to Network Applications

4.4.1 The Problem

End-to-end QoS support is required to support multimedia communications in the Internet. Hence, mechanisms to enable end applications to request desired QoS levels from underlying networks is important.

In contrast, there exists a large pool of non-QoS aware applications (hereafter referred to as legacy applications) that are unable to exploit and benefit from the QoS support available in networks. The technology to support QoS in networks is not yet fully mature. Thus, developing an application to interact with a specific QoS protocol carries the danger that the application may become obsolete if the QoS protocol is modified or superseded.

Different APIs like the RAPI for RSVP, the QoS API from the Internet2 community [151], and the generic QoS API integrated in WinSock2 from Microsoft [152] have been developed to enable applications to request reservations from QoS provisioned networks. However, these APIs are mainly intended for application developers rather than end users. The task of providing QoS support to network applications has been studied by other research groups [153–156]. Many of the proposals were based on modifying the underlying operating system for QoS support [155] or to use signalling protocol specific software modules at user nodes [156].

It will not be easy to modify existing end applications or OS to integrate QoS support. Furthermore, such an approach will not be readily realisable and scalable in large networks. Additionally, developing an application to interact with a specific QoS protocol carries the danger that the application may become obsolete if the QoS protocol is modified or superseded.

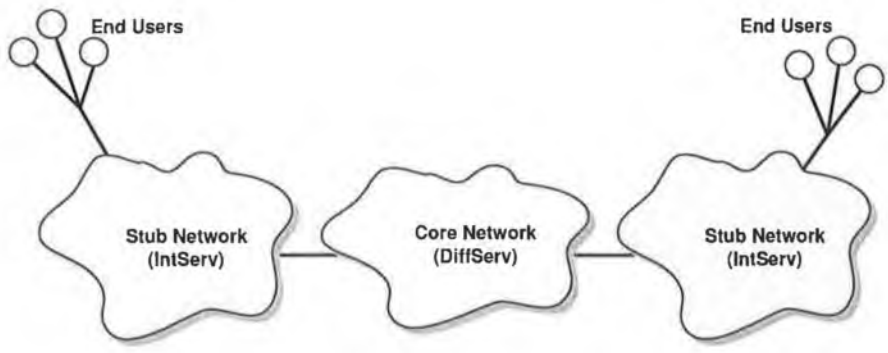


Figure 4.10: End-to-End QoS Model: IntServ and DiffServ

4.4.2 QoS Support using Netlets

Due to the above problems, a remote service invocation method is proposed by the Netlets approach to provide signalling support for end applications. Furthermore, by developing solutions that are independent of the underlying signalling protocols, it will be possible to transparently migrate with changes to the underlying QoS schemes.

The reference network which I have used to describe the approach for providing QoS support to legacy applications is based on the combined QoS of Intserv and DiffServ model (Fig. 4.10). We use Netlets to enable QoS support to end applications operating over such a network environment.

Below, the mechanism to couple legacy network applications with QoS support features using Netlets is described. The QoS support to end applications are based on user requests to a manager Netlet node present in the stub network. This manager node processes and coordinate the QoS-support requests from end users and also performs the deployment of Netlet services for QoS signalling. The deployed Netlet services interact with the IntServ based stub network on behalf of end applications to provide an end-to-end QoS support. The details of the complete process involved to enable signalling support are presented below.

In order to invoke QoS signalling support from other than the end host running the application, the primary tasks involved are: (a) identify the occurrence

of the flow belonging to the session requesting QoS support; and (b) to know the lifetime for which the signalling service has to be in place. Hence flow monitoring and signalling support services are mandatory. In the Netlets approach these services are handled by Netlet component themselves.

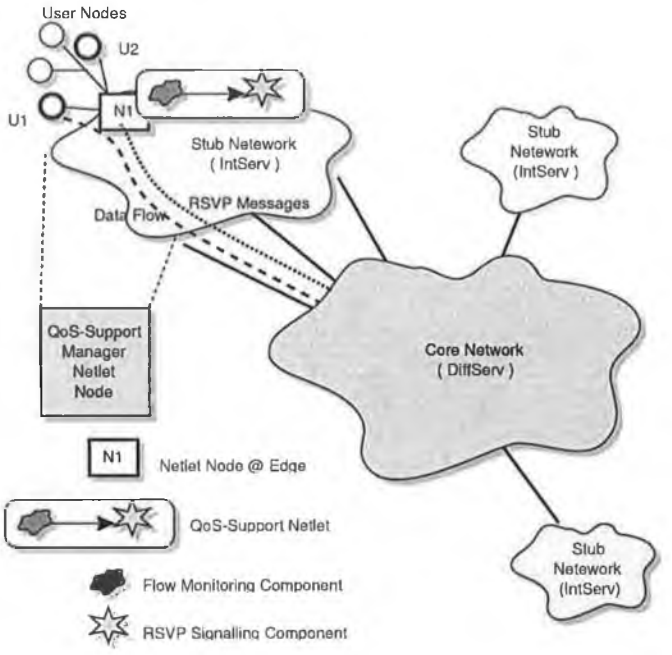


Figure 4.11: QoS Support Using Netlets

The QoS-Support Manager Netlet Node: The general framework of the Netlets based approach to enable QoS support to legacy applications in the Internet is shown in Fig. 4.11. The manager node functions to interact with and receive QoS-support requests from end users through a web-based utility. This node is assumed to be a well known node in the network. The QoS-support request generated by the web-based utility on behalf of the session requiring service contains the following information: the sender and destination addresses, the *application type* (data, multimedia, groupware) and its *operational mode* (either as a sender, receiver or both). The information about the *application type* helps the manager node to make an initial estimate of the source's traffic characteristics. In the case of the IntServ network, this information will allow an initial choice of the TSpec

parameters. The *operational mode* of the application indicates the signalling features required to support the application's traffic. A prototype version of the web-based utility used in the implementation is shown in Fig. 4.12.

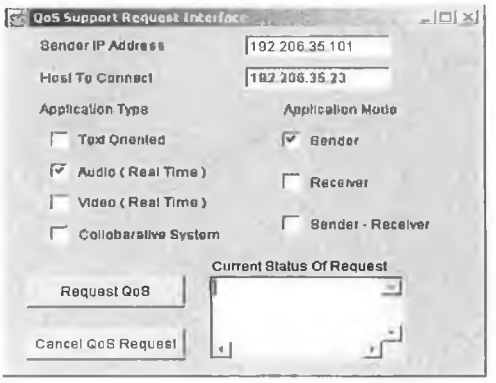


Figure 4.12: QoS Support Request Interface

The manager node contains and deploys (based on user requests) the Netlet components for providing QoS support. On receiving a QoS-support request for an end application (for example from U1 as in Fig. 4.11), the manager node runs a trace-path program to identify the edge node of the stub network to which the requesting end host is attached (Node N1 as is Fig. 4.11). On identifying the connecting edge node, the manager node deploys a QoS-Support Netlet (discussed in the next section) at that edge node to enable QoS signalling for the requesting end application. Here I assume all the edge nodes present in the stub network are Netlet enabled active nodes. A case for the presence of non-active segments along the stub network edge is discussed in later part of section 5.4.3.

The purpose of deploying the Netlet component at the edge node is to identify the occurrence of the flow pertaining to the session requesting signalling support and to provide online traffic modelling of the flow. As in the case of an IntServ based network, traffic modelling allows the accurate calculation of the TSpec parameters. Source traffic in the Internet typically follows a Variable Bit Rate (VBR) pattern and thus continuous traffic estimation of the flow will have to be performed for accurate resource reservation.

A QoS-Support Information Table (QiT) is used at the manager node to manage the Netlet components operating in the stub network. This table records the details of the end users requesting service and the address of the Netlet edge nodes at which the Netlets corresponding to the requests resides to provide QoS-Support.

To avoid multiple copies of the same Netlet being present at a node to serve individual flows, the QoS supporting Netlet can be designed to handle multiple flows simultaneously. For example, if in the case of another user U2 (see Fig. 4.11), connected to the edge node N1 (N1 already hosts Netlet service for U1) requires QoS support, the manager node requests additional service from the Netlet at N1 (based on information available in QiT) instead of deploying a new Netlet service at N1.

The QoS-Support Netlet: This Netlet encapsulates flow monitoring and signalling components for QoS service support. On initialisation, the manager node feeds this Netlet with the session details (obtained through the web-utility) of the flow for which the QoS support has to be enabled. This input also includes an initial estimate of the TSpec parameters. This estimate aids the Netlet in starting the reservations immediately and also allows a short convergence period during which the traffic measurement process of the signalling component gets stabilised. On migration to the edge node to which the end host is attached, this Netlet autonomously starts the monitoring component to identify flows belonging to the session requesting QoS signalling. The flow monitoring is based on the pair of source and destination addresses, communication ports and protocols used.

Based on the flow information collected by the monitor component, the FilterSpec parameter of an application's packet streams can be accurately obtained. The monitor component triggers the signalling component on occurrence of the flow pertaining to the requesting session. The signalling component then uses the

FilterSpec information along with the TSpec parameters of the application's flow to reserve resources. Traffic measurement capabilities present in the signalling Netlet component allows the application's source traffic pattern to modelled on-line.

The operational support offered by the signalling component is based on the *operational mode* of the application (obtained through the web-based utility). If the application is either sending/receiving packets into the network as in the case of a video server/media player receiving video packets, then the key function of the signalling element is to send PATH/RESV messages respectively before timeout periods to confirm reservations. In the case of an application being both sender and receiver as in the case of video conferencing and collaborative system applications the signalling component performs both reservation requests and actual reservations in the network.

The QoS-Support Netlet is designed to handle multiple flows simultaneously. This reduces the need for multiple Netlet services required to be present at a single node for serving individual flows. The ability of the Netlet to clone and relocate itself to a new node avoids the need for the manager node to host and deploy services in a centralised fashion.

4.4.3 Benefits of Using Netlet Services

Introducing New Network Services: Introducing new services in the Netlets network is performed dynamically. For example emergence of a new/modified version of the signalling protocol for the IntServ model will only require removing the existing Netlet services and introducing new services which implement that protocol.

Migratory Path: The Netlets approach to provide QoS support to legacy applications offers a reliable and flexible approach without actually making the end applications QoS aware. This approach can be used to migrate from the current

application model which either provides restricted or no QoS support to a model which provides QoS support of choice on demand.

4.4.4 Remarks

I described a novel approach based on Netlets to transparently retrofit QoS support to legacy network applications. The Netlets approach is not restricted to a single QoS model or signalling protocol. Thus it may be continue to be used even if the QoS support provided by the underlying network changes.

4.5 Summary

We proposed solutions using Netlets:

- to overcome the problem of reservation gaps;
- to support transparent and scalable client-side server selection solution in the Internet;
- to build a scalable and reliable multicast protocol that accommodates heterogeneous receiver terminals; and
- to retrofit QoS support for existing and emerging network applications.

In addition, the Netlet based solutions demonstrated a new class of networking services and their benefits: They are: (i) services that represent users in the network – those services that are able to represent users within the network (e.g. Netlet director and Hot Spot Delegate) ; (ii) self-organisable services – those Netlet based services that are able to self-organise based on network events (e.g. reservation gap Netlets); (iii) services that populate on demand (e.g. RepN services in MENU). These services must meet certain performance requirements if they are to be of practical value. Their performance is assessed in Chapter 5,

where I present results based on evaluations of the individual applications and also conduct measurements of the performance of elements of the Netlets architecture to evaluate the wider applicability of the Netlet prototype.

Chapter 5

Evaluation of Proposed Applications

This chapter presents results from experiments that were carried out to evaluate the set of applications described in Chapter 4. In order to evaluate the practicality of the Netlet prototype for a wider set of applications than those discussed in Chapter 4, I conducted generic tests to analyse its performance characteristics and that of various service deployment schemes. Finally, issues relating to the practical deployment of Netlets in the Internet are discussed.

5.1 Robust Reservation Support using Netlets

In this section, we evaluate the solution to the problem of reservation gaps described in section 4.1. We present two distinct sets of experiments to study the problem of reservation gaps. In the first set, using a laboratory testbed setup, I compare operational characteristics of Q-flows when traversing non-managed and Netlet managed reservation gaps. In the second set of experiments, using simulations I[1] present a comparative analysis between the most reliable – shortest path (MR-S), the shortest – most reliable (S-MR) routing algorithms and the traditional shortest path algorithms.

[1]Kalaiarul Dharmalingam and Karol Kowalik [8].

5.1.1 Unmanaged Vs Netlet Managed Gap

Its expected that, in a heterogenous environment like the Internet, a non zero number of reservation gaps will occur even when using routing algorithms such as *MR-S* and *SR-M* to select paths for the Q-flows. Hence it is desirable to study the characteristics of Q-flows when traversing unmanaged and Netlet managed reservation gaps.

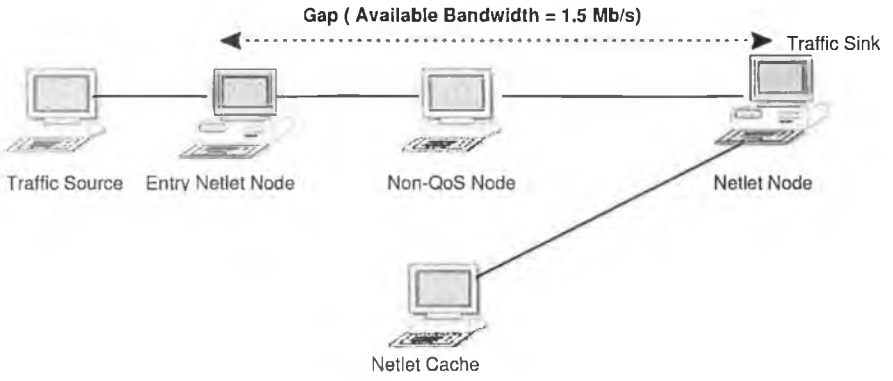


Figure 5.1: Experimental Setup for Robust Reservation Support

I use the experimental setup shown in Fig. 5.1 for this purpose. In the first set of experiment, I evaluate the case when no gap management support is available. We use three UDP based Q-flows to traverse the non-QoS link, referred to as L, of capacity 1.5Mb/sec. The UDP traffic generator is a constant bit rate (CBR) source with exponentially distributed on and off periods. We define Q-Flow 1, and Q-Flow 2 with bandwidth requirement ≈0.4Mb/sec and 0.9Mb/sec respectively. When a new Q-flow, Q-Flow 3, with bandwidth requirement ≈1Mb/sec, greater than the available resources, entered the reservation gap, the link was overloaded. This causes degradation to all Q-flows as shown in Fig. 5.2. All three flows suffer heavy packet losses and the network resources are inefficiently utilised.

Next, I evaluate for the case when gap is managed. I designed a Netlet, referred to as gap management Netlet, for this purpose. The role of this Netlet is to

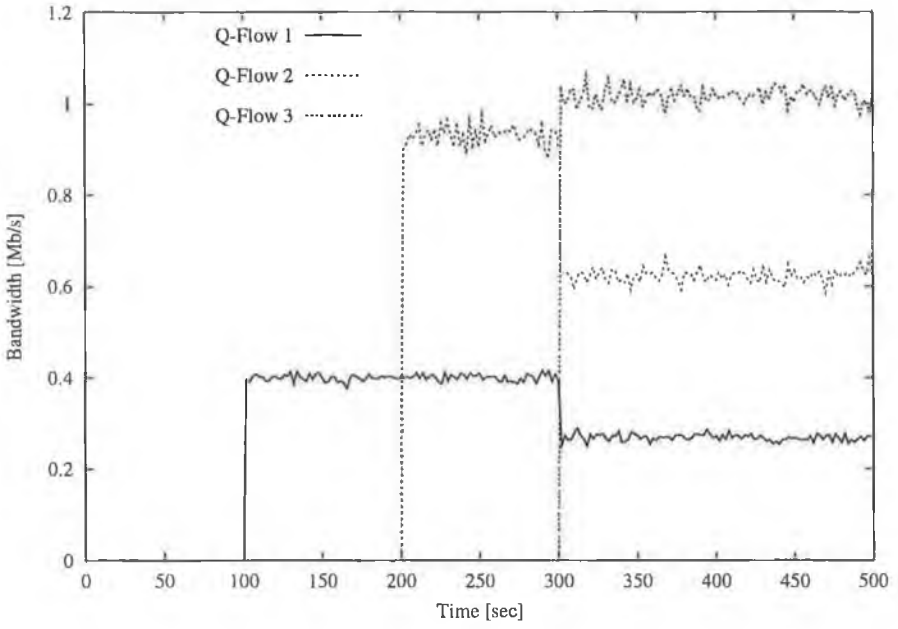


Figure 5.2: Service Degradation For QoS Flows across the Gap

manage the reservation gap and to reject those flows that request more than the available gap bandwidth. For the purpose of simplicity, the effective gap to be managed bandwidth is stored within the NRE, 1Mb/s in this case.

The gap management Netlet by consulting the NRE is able to find the maximum effective gap bandwidth that had to be managed. The Linux port of the RSVP[2] package is used for this testing. The Packet Communication Engine (PCE) at the Netlet node is configured to capture RSVP packets (i.e. packets with protocol number 46). A simple C program was written, which uses the RSVP API to request a reservation and then starts the UDP traffic generator.

At t≈100 the C program corresponding to flow Q-Flow 1(Fig. 5.3) is initiated. The Netlet Management Engine of the exit node (the traffic sink in Fig. 5.1) on identifying a gap downloads the gap monitoring Netlet from a neighbour Netlet cache and instantiates it. The gap monitoring Netlet on obtaining the address of the gap entry node from the Netlet Management Engine (NME) sent a clone of itself to the entry node.

[2]RSVP Port Under Linux, http://www.isi.edu/div7/rsvp/release.html

At t≈200 the C program corresponding to flow Q-Flow 2 is initiated. The entry gap Netlet on receiving the request for reservation, compares them to the available resources (available 0.6Mb/s). Since there are enough resources available to support the reservation, the RSVP packets are forwarded towards the destination node. Furthermore, the entry Netlet updates the value of the available gap bandwidth to zero.

When the third flow Q-Flow 3 sends an upstream reservation request, the gap management Netlet, drops the request. It also sends an error message to the node, which requests the reservation notifying it of the non-availability of resources across the non-QoS link, L (in Fig. 5.3, at t≈300). In this case, Q-Flow 3 does not interfere with the existing Q-flows (Q-Flow 1, Q-Flow 2) which continue to receive the requested QoS (Fig. 5.3) and the network resources are thus efficiently utilised.

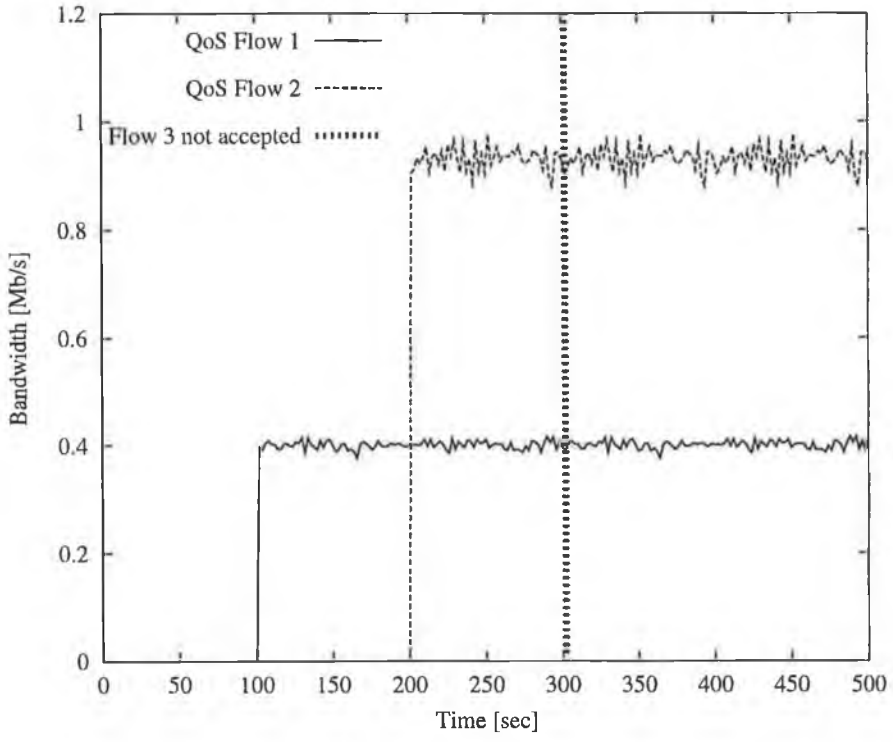


Figure 5.3: Netlet Managed Reservation Gap

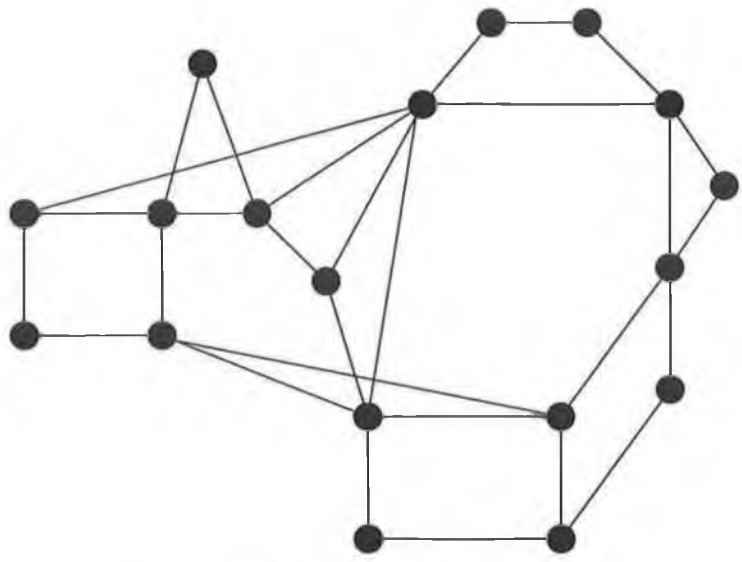


Figure 5.4: The ISP topology

5.1.2 Routing Enhancements

Network Model: I evaluated the performance of three routing approaches (SP, MR-S and S-MR) described in Section 4.1.6 with the so-called ISP topology [157] shown in Figure 5.4. In general, the network topology is assumed to consist of N nodes connected using L bidirectional links each with capacity C (for the ISP topology we have used $N = 18$, $L = 30$ and $C = 20$). We also assume that within the network there are N^Q Q-nodes and N^{nQ} non-Q nodes (where: $N^Q + N^{nQ} = N$).

The requests arrive at each node independently according to a Poisson distribution with rate λ and have exponentially distributed holding times with mean value $1/\mu$. The requested amount of bandwidth is uniformly distributed over the interval: $[64kb/s, 6Mb/s]$, with mean value $B = 3.32Mb/s$. If traffic is generated by N^s source nodes, it produces the network offered load [158]: $\rho = \lambda N^s B h'/\mu LC$, where h' is the average shortest path distance between nodes, calculated over all source-destination pairs (for the ISP topology: $h' = 2.36$ if $N^s = 18$). In our experiment we have used a mean connection holding time of 60sec and choose λ to produce the required offered load in the network.

Performance metrics: In our simulations we assumed that when a new request arrives it can receive one of two responses when path monitoring is used:

- accept - if there are enough resources along the chosen and monitored path;
- reject - if resources along the chosen and monitored path cannot accommodate the new request.

However if path monitoring is not present (as in the existing Internet) there may be a third outcome, viz failure.

fail - if the decision was to accept a connection on the path, but the path failed to provide the required QoS level. This occurs if the user terminates the connection because QoS level does not conform to the requested quality.

We have assumed that the lifetime of failed connections is exponentially distributed with a mean value equal to half of the mean value of a standard connection ($1/2\mu$). We will explain why such a value was chosen in Section 5.1.3.

We[3] have used the following metrics to compare the performance of the three routing approaches (SP, MR-S, S-MR): the *call blocking rate* – defined as:

$$\text{call blocking rate} = \frac{\text{number of (rejected + failed) requests}}{\text{number of arrived requests}}$$

which is used to calculate the probability of rejecting the new request; the *average path length* – defined as:

$$\text{avg. path length} = \frac{\sum \text{path length of accepted connections}}{\text{number of accepted connections}}$$

used as an indicator of resource consumption when comparing the algorithm to algorithms which limit the hop count; the *installation cost* – defined as:

$$\text{installation cost} = \frac{\text{number of monitoring points}}{\text{number of monitored connections}}$$

[3]Results were obtained jointly with Karol Kowalik.

which is used to calculate the cost of installing active services along the path; and the *reliability* – defined as:

$$\text{reliability} = \frac{\text{number of QoS aware nodes along the path}}{\text{number of total nodes along the path}}$$

(a path is assumed to be more reliable if it has a higher ratio of Q-nodes).

5.1.3 Simulation Results

The two proposed routing algorithms (MR-S and S-MR) aim to improve the reliability of routing protocols in a heterogenous environment. Hence for the given network with the ISP topology we have randomly increased the number of Q-nodes starting from a network which does not provide any QoS support ($N^Q = 0$ and $N^{nQ} = N$) until we have reached the fully QoS supportive network ($N^Q = N$ and $N^{nQ} = 0$) – we call this a cycle of simulation. In our experiment each cycle of simulation was repeated 500 times and we present the average results below. As shown in Fig. 5.5 when path monitoring is not supported and the shortest path (SP) is chosen, the blocking probability is quite high for networks with a small number of Q-nodes. This is caused by connections being setup despite there not being enough resources to accommodate them and which fail after establishment. This type of failed connection uses resources unnecessarily and so blocks other potential connections. We can only reduce the blocking probability when using SP by increasing the number of Q-nodes. We used $1/a\mu$ as the mean holding time of failed connections. We assumed that $a > 1$ because we expect that a failed connection will be terminated earlier than had it been successful. In our simulations we used $a = 2$. Using other values of a, which are greater than one, changes the blocking probability but this remains a monotonic decreasing function of the number of Q-nodes.

If a path monitoring mechanism is employed in conjunction with routing al-

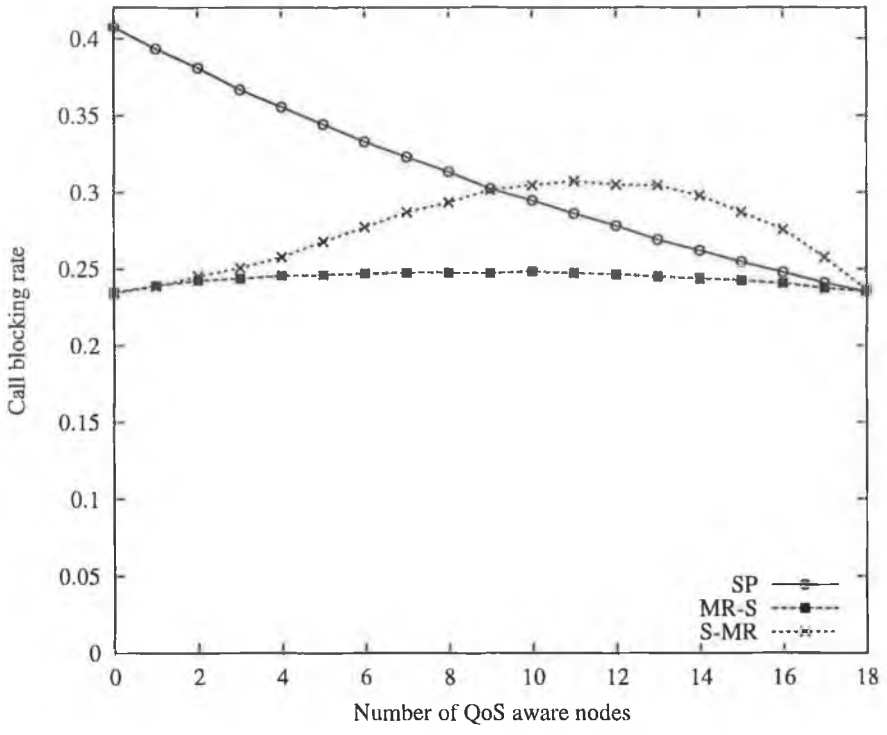


Figure 5.5: Call blocking probability of SP, MR-S and S-MR under increasing number of QoS aware nodes

gorithms which select more reliable paths (as in the MR-S and S-MR curves in Figure 5.5) the blocking probability is reduced. By using monitoring we prevent situations arising where, due to a lack of accurate reservation information, connections are established over links with insufficient resources. The MR-S algorithm, which chooses the most reliable path from the set of shortest paths, shows the extent to which blocking probability can be reduced by path monitoring. The S-MR algorithm also reduces blocking probability, when the number of Q-nodes is small. However when the number of Q-nodes is more than half of the total number of nodes, it generates a high blocking rate, because of the use of non shortest paths which consume extra resources. This is clearly seen by comparing Figures 5.6 and 5.5. This also confirms the findings of other researchers [158], that algorithms limiting hop count produce a lower blocking probability. The average path length of SP (the bottom curve in Figure 5.6) grows slightly when the num-

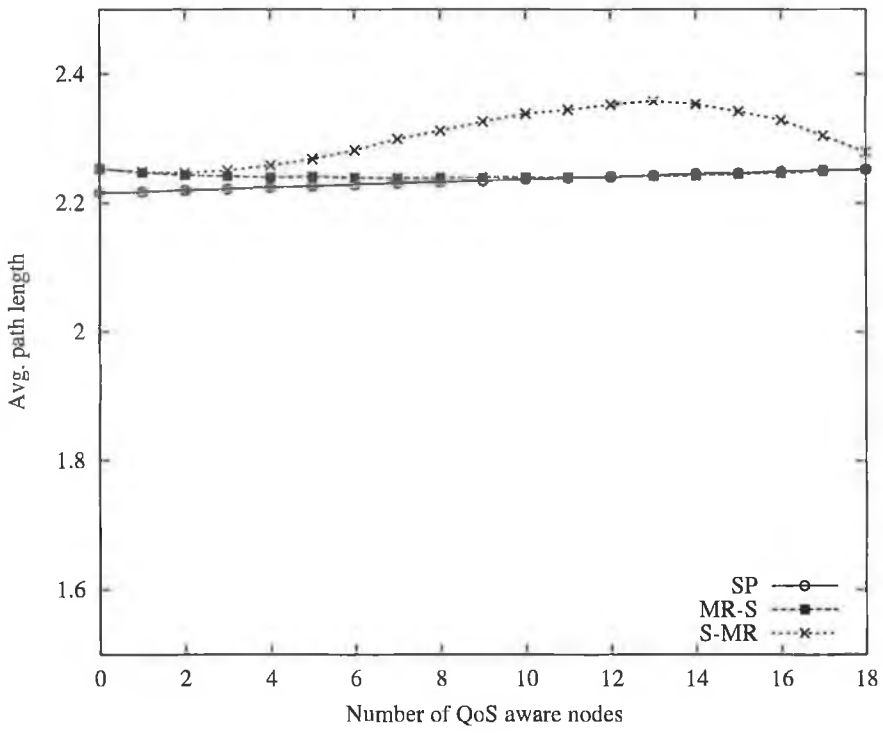


Figure 5.6: Average length of the path chosen by SP, MR-S and S-MR

ber of Q-nodes increases. Clearly, when information about resources available in the network is not provided, setup of connections requiring only a few hops is more likely to be successful than of connections where the source and destination nodes are further apart.

Although S-MR produces a higher blocking probability than MR-S, it is more reliable when compared with the SP and MR-S algorithms (see Figure 5.7). The path reliability shown in Figure 5.7 does not differ much for each of the algorithms. This is due to the fact that Q-Nodes are selected randomly and there are not many alternative paths featured by ISP topology. When Q-Nodes are grouped and not dispersed, the S-MR offers significant improvement in reliability over other approaches. This suggests that the S-MR algorithm should be used only for connections requiring high reliability and producing higher revenue. Other flows should be processed using MR-S.

When evaluating the cost of installing the monitoring mechanisms in Fig-

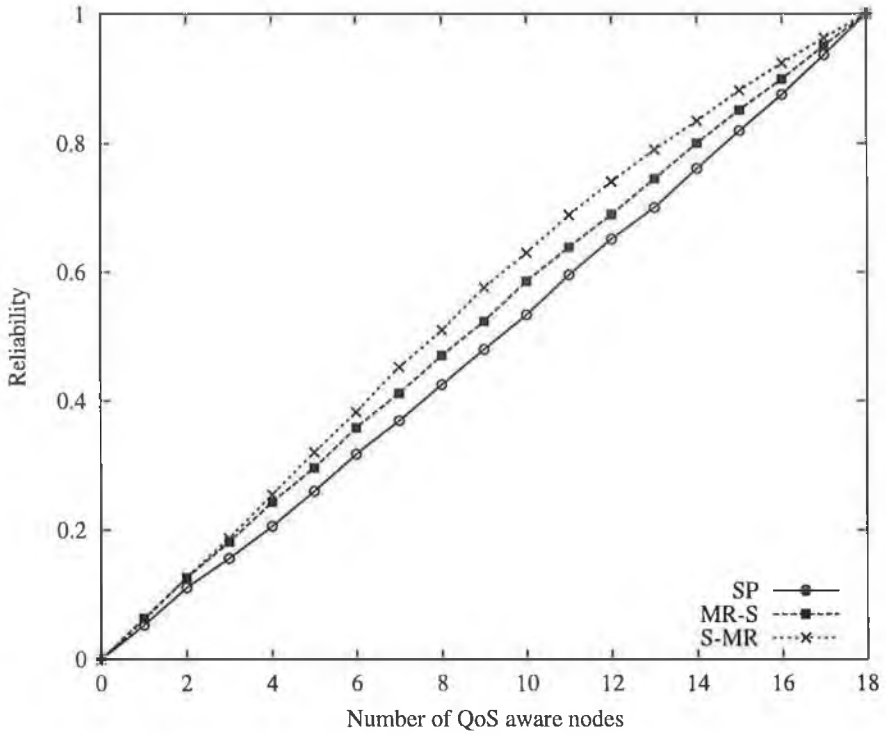


Figure 5.7: Reliability of routing decisions of SP, MR-S and S-MR

ure 5.8 we can see that the cost of using monitoring for S-MR is comparable with that for MR-S (when the number of Q-nodes is less than half of the total number of nodes) or is even lower (if the number of Q-node is greater than half the total number of nodes). So if a network administrator decides that the computational cost of monitoring non-Q segments is excessive, he could use S-MR even if it produces a higher blocking probability than MR-S.

5.1.4 Remarks

I presented two distinct sets of experiments to study the problem of reservation gaps. First, I compared operational characteristics of Q-flows when traversing non-managed and Netlet managed reservation gaps. Next, I presented a comparative analysis between the most reliable – shortest path (MR-S), the shortest – most reliable (S-MR) and the traditional shortest path algorithms. The technique

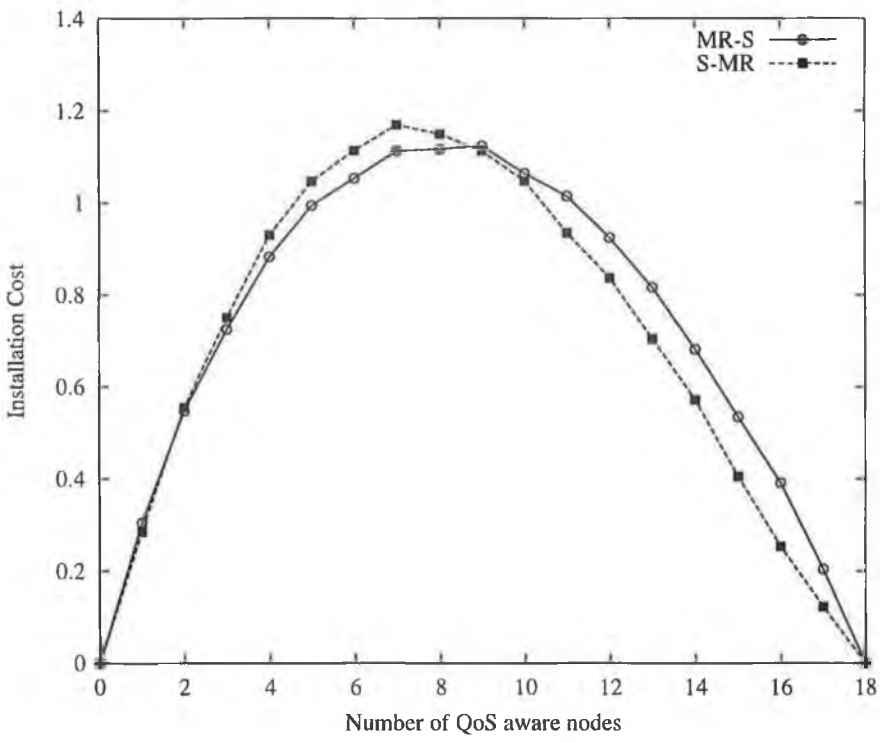


Figure 5.8: Cost of path monitoring

described here makes it possible to deploy applications in the network which have quite hard QoS guarantee requirements, even when a significant number of network nodes support only best-effort service. Such techniques will be of critical importance in ensuring the graceful transition of the Internet from a best-effort service model to a service model featuring QoS guarantees.

5.2 Server Selection using Netlets

In this section, I present an evaluation of the client-side server selection mechanism proposed using director Netlet services (in section 4.2). The evaluation presented below includes results from tests performed in a LAN environment and on the Internet. The first set of experiments evaluates the working of a director Netlet to act as switching agent to perform load distribution across servers. The second set of experiments, evaluates a range of selection metrics that can be

used by director services to support server selection in the Internet.

5.2.1 Client Perceived Service Response Time

The service response time perceived by a client can be formulated as:

$$ServiceTime = T_{Locate} + T_{Connect} + T_{Serve} \tag{5.1}$$

where, T_{Locate} refers to the time taken to locate a server; $T_{Connect}$ is the time required to establish the connection; and T_{Serve} is the remaining time taken to serve the request.

The T_{Locate} and $T_{Connect}$ components are dependent on the prevailing network and server conditions. The T_{Serve} component is largely dependent on the request type but also depends on the server load. Hence, selection metrics that use server load and network parameters to make server selection decisions will be able to control the effect of these components on the total service response time perceived by the client. I compare the performance of the following server selection metrics when employed by the director services: (i) server load; (ii) network latency; (iii) random selection and (iii) end-to-end processing delay.

5.2.2 Server Load Distribution

The first set of experiments included: (i) the implementation of load distribution across servers using director Netlet services; and (ii) a study of the impact of server load on client perceived service response time.

The experimental setup used for the analysis is shown in Fig. 5.9. The working of the experimental setup is as follows. Client program at node C initiates connection requests to the anycast address of the server group, A. Furthermore, the node C has Netlet node, ND, as its gateway node. The director service is instantiated at ND and a packet capture filter is configured with the Packet Com-

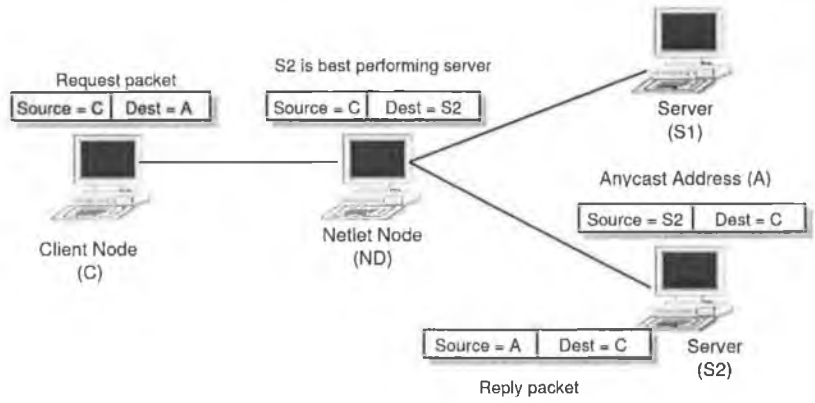


Figure 5.9: Experimental Setup for Load Based Server Selection

munication Engine (PCE) of node ND. Note the filter specifies to capture TCP SYN packets destined to address A.

When the PCE of the Netlet node ND receives such packets, it captures and hands them over to the director Netlet. This Netlet then changes the destination address of the TCP SYN packet to the address of the best performing server (either to S1 or S2 based on measurement results). Note a C program was written and installed at both servers, so as to capture and revert back the source address of the HTTP session packets from the server's address (either S1/S2) to the anycast address, A. This is because the TCP control block at client node will have the session destination address A. Hence, to establish connections, the destination address will have to be maintained.

5.2.3 Experiments

The below set of experiments were carried out in a LAN environment with a pair of Apache servers [159] (S_1, S_2) running on Linux machines. Server S_1 is configured the closest to the client node (a single HOP away) while server S_2 is configured with 2 HOPS as the distance metric.

These servers are configured to accept a maximum of 150 connections simultaneously. The mod-status module present in the Apache server is configured to

monitor the load condition on the servers. Httperf [160], a HTTP traffic generation tool is used to generate background traffic on servers. Client requests to servers are modelled with exponential inter-arrival times. Here, a Java program was written to act as the web client.

The goal the director service has to accomplish is: *to route client requests to the best performing server based on load conditions (obtained using the mod-status module of apache), thus achieving load distribution across the servers.* The maximum load threshold at the servers is defined as 80%. The director services works with this value for server selection.

The httperf tool generates background traffic to S_1, constituting around 90% load on the server for the first 350 seconds (see Fig. 5.10). During this period of time, the director service routes requests to server S_2. When the background traffic is removed from server S_1, the Netlet service directs requests to the closest best performing server, S_1. This corresponds to the period from 350 to 750 seconds in Fig. 5.10. When the background traffic is introduced back on server S_1, the director service routes requests to S_2, thus accomplishing load distribution.

To study the impact of server load on service response time, average download latency for files from the two servers are analysed. File sizes that are used in the tests vary from 500K to 5000K. Files are downloaded from server S_1 when it is operating at 80% load and when server S_2 is having 40% load imposed. This corresponds to load conditions at different instants of time in Fig. 5.10. The average download latency experienced for each file at both servers are shown in Fig. 5.11.

The average download latency offered by server S_2 is 2 to 3 times less than that of the closest server S_1. Thus, we can conclude that the server load affects the response time perceived by clients. Furthermore, approaches (e.g. [12]) based on locating closest server replicas nodes for serving client requests thus does not always provide an accurate server selection technique.

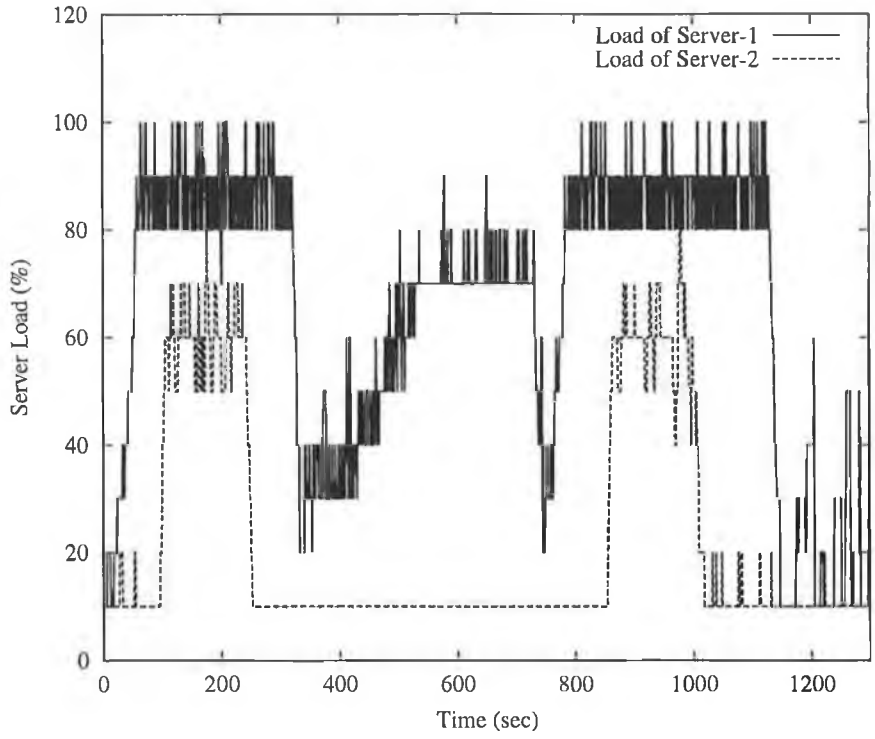


Figure 5.10: Load Based Server Selection

5.2.4 Server Selection in the Internet

The experimental setup that I used for this test is shown in Fig. 5.12. The director Netlet is downloaded and instantiated from the neighbour Netlet cache node. The web client that is responsible for generating requests and the director Netlet are located on the same node.

The working of the director service is as follows. The director service performs measurements and records the address of the best performing set of servers to a file, referred to as the *weather* file. Using these results, the web client then establishes connections to the appropriate server. Note to implement the idea of changing destination addresses 5.2.2 the server end would have to be modified. However, such a facility is not available, when working with servers on the Internet.

For this experiment, I used a set of 10 mirror servers (www.kernel.org) [161]

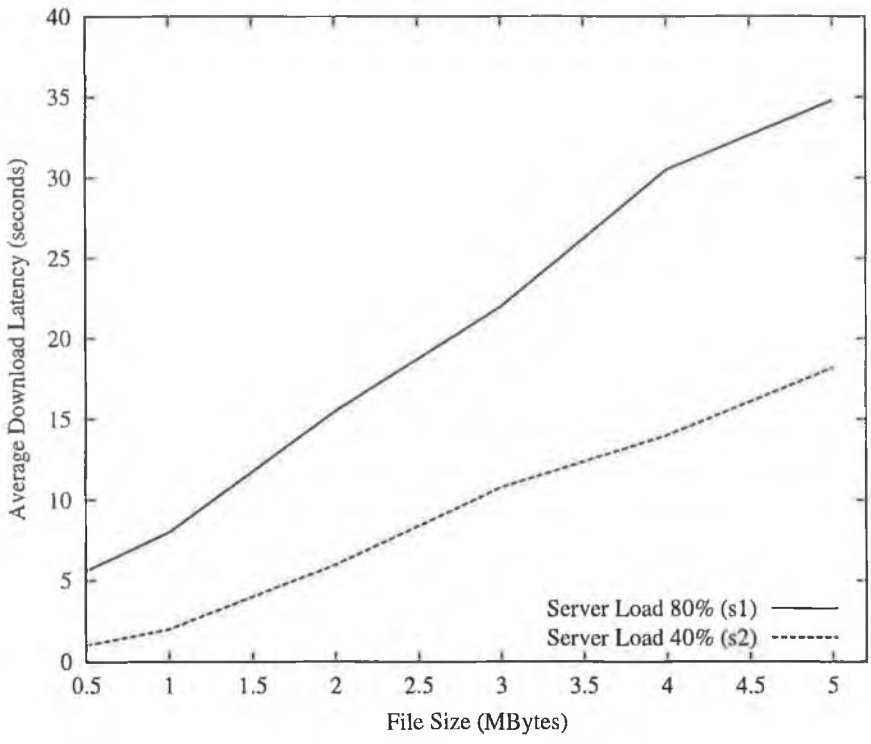


Figure 5.11: Impact of Server Load on Service Response

present at different geographical locations in the Internet. File sizes used for testing the average download latencies experienced by clients varied from 500K to 15000K. The total set of measurements spanned a 10 day period at different times of the day so as to minimise effects of caching and time-of-day effects. In this set of experiments, I evaluate three different metrics for server selection in the Internet.

Random Selection: For this metric, the director services implements the random server selection strategy popularly followed in the Internet. Random number generators are used to decide the server to be selected from the replica set. Average download latency for each file (500K to 15000K) is recorded (see Fig. 5.13).

Network Latency: In the second server selection technique, the director services uses network latency as the parameter for deciding the optimum performing server. The average round-trip time (RTT) is measured for each server by the Netlet service. This measurement is carried out using ping probes to each server.

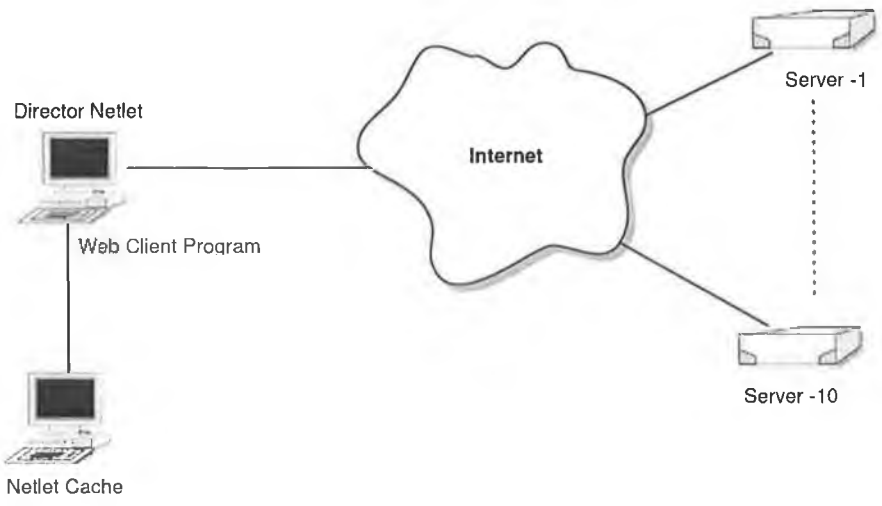


Figure 5.12: Experimental Setup for Server Selection in the Internet

The selection decision at the Netlet service is either made on past probe measurement results or on new measurements that are made prior to assigning the requests to a server. The timeout period for past measurement results is assigned as 180 seconds. The timeout value is arbitrarily chosen to reduce frequent probing. The probability of selecting a server, S_j, from a replicated set of N server replicas is calculated using the following equation:

$$Prob\,(s_j) = \frac{1/RTT_{S_j}}{\sum_{j=0}^{N} 1/RTT_{S_j}} \tag{5.2}$$

where RTT_{s_j} is the average round-trip time that corresponds to server S_j from the director service.

Based on the measurements, the server with the highest probability is selected. The average download latency for each file is recorded (see Fig. 5.13).

End-to-End Latency: The selection metric based on network latency does not account for the load condition at server nodes. This is because the ping responses from servers are handled by server daemons other than the web server daemons. Applications that are both CPU intensive and delay sensitive will require both server and network load parameters to be involved in deciding the best perform-

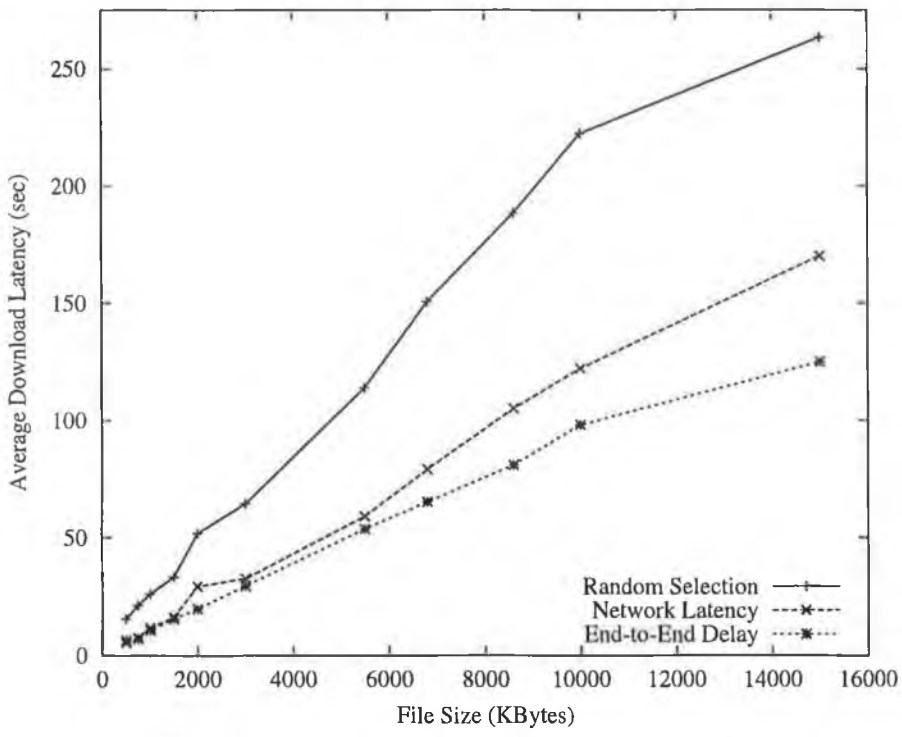


Figure 5.13: Comparison of Server Selection Metrics

ing server. A solution to support such decisions will be to check the end-to-end latency perceived by the client. The end-to-end latency, L_{S_j}, can be formulated as

$$L_{S_j} = RTT_{s_j} + Pdelay_{S_j} \tag{5.3}$$

where, RTT_{s_j} is the average round-trip time to server S_j from the Netlet service and $Pdelay_{S_j}$ is the request processing delay at the server. The end-to-end latency can be measured by downloading a small test file from all server replicas.

A 100K file is used to measure the end-to-end latency in the experiments. The timeout period for past measurement results as 180 seconds. The probability of selecting a server, S_j, from a replicated set of N server replicas is calculated using the following equation:

$$Prob\ (s_j) = \frac{1/L_{S_j}}{\sum_{j=0}^{N} 1/L_{S_j}} \tag{5.4}$$

The average download latency for each file from the group is recorded (see Fig. 5.13).

Metric Comparison: The end-to-end latency technique performs the best among all the three schemes discussed (see Fig.5.13). This finding is consistent with results presented in [136]. The random selection technique popularly followed in the Internet offered the worst performance. The average download latency offered by this technique was three times more than the end-to-end latency and almost twice that of the network latency based approach. The technique of downloading small test files to measure end-to-end latency will not scale for servers containing large set of replicas in the Internet. A scalable approach as described in section 4.2.7 can be adopted.

5.2.5 Remarks

I presented the evaluation of the client-side server selection mechanism I proposed using Netlet director services. By using Netlets, service decision points were able to be dynamically deployed in the network at locations where they can most efficiently serve a large number of clients. Overall, this approach demonstrated the versatility of implementing server selection algorithms that can work on the client-side of the network.

5.3 Large Scale Deployment of MENU

In this section I evaluate the large scale deployment of the MENU protocol (section 4.3). Furthermore, I also evaluate the benefits of employing MENU over the other existing multicast protocols. Simulations are used to carry out the analysis.

5.3.1 Experiments

Netlet nodes are required to be present in the Internet, in order to host the Replication Netlet (RepN) services, which provide packet duplication support in the MENU protocol. However, due to the large scale nature and heterogeneity of the Internet, such support can only be integrated gradually. Under such circumstances, MENU like protocols will go through phases of partial deployment to ubiquitous availability.

Here, I study through simulations the effect of incremental deployment of Netlet node support in the Internet on MENU. I evaluate this based on: (i) the gain (e.g. reduction in bandwidth) offered when using MENU based multicast communication services over unicast; (ii) the packet redundancy level, referred to as the link stress [162], experienced by the network nodes when working with partial deployment of Netlet nodes in the network; and (iii) the forwarding state saving achieved by MENU in comparison to existing multicast protocols. Finally, I study the minimisation in error recovery delays when supporting data caches at Netlet nodes in the network.

5.3.2 Network Model

This analysis is performed using core-stub network topologies generated using the GT-ITM [163] package. The topologies for the study have 20 nodes per stub domain and 10 nodes per core domain. The total number of router nodes present in the generated topologies are 500. Furthermore, 20 user nodes are added at random to each stub domain in the network. Note that at most a single stub router had only two receivers assigned. For the purpose of illustration, the topology I use for testing the effects of mean hop count (12.5) against multicast gain is presented in Fig. 5.14.

It is assumed that each stub domain represented a potential community of interest for the traffic source and all receivers subscribe for the multicast session.

Figure 5.14: Network Topology used for Multicast Gain Measurement

For each simulation cycle, the traffic source is selected randomly from one of the stub domains. I use different randomisation seeds during each simulation cycle for assigning network routers as "Netlet Supportive". I present the results averaged across 25 simulations.

Efficiency of employing MENU based multicast communication over unicast was evaluated using the gain metric defined in [164]. The multicast gain in refer-

ence to bandwidth saving is defined as:

$$\delta = 1 - \frac{L_m}{L_u} \tag{5.5}$$

where L_m denotes the total number of multicast links in the distribution tree and L_u is sum of unicast hops. δ represents the percentage gain in multicast efficiency over unicast. For δ approaching zero, multicast offers no gain over unicast communication. As δ increases (to a maximum value of $\delta = 1$) multicast communication offers bandwidth savings over unicast.

5.3.3 Results

Active Stub and Non-Active Core Domains

This experiment is carried out to study the multicast gain when stub network domains hosted Netlet nodes i.e. where there were stateful stub domains and stateless core domains. The effect of incremental deployment on MENU is assessed by varying the number of Netlet nodes available within the stub networks. In the experiment, the ratio of Netlet nodes available in each stub network is varied from 0% to 100% in steps of 10. The traffic delivery tree is constructed as described in section 4.3.8. For each deployment ratio, the number of multicast hops to unicast hops when serving all the receivers in the network is recorded. This experiment is repeated for different mean path lengths (between the source and receiver nodes). This allowed us to evaluate the impact the hot spot nodes have on the multicast gain.

Fig. 5.15 shows the results of this analysis. The case of 1 Netlet node per stub domain denotes the existence of a single Hot Spot Node for each community of interest. In the MENU model this will result in individual unicast connections being setup from source to each HSN and from each HSN to all its end users. With only the HSN being present, the gain varied from 30% to 73% for mean path

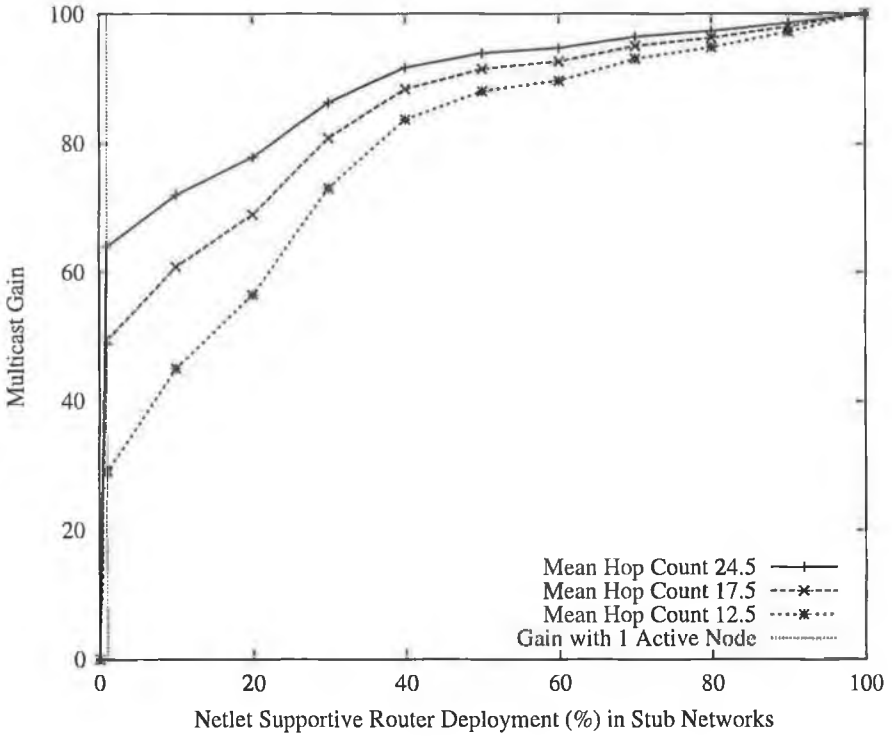


Figure 5.15: Multicast Gain with Active Stub Networks

lengths from 12.5 to 23.5 respectively. This is due to fact that all receivers from a community were served from the local HSN, which in turn communicated with the traffic source. The reason for the increase in gain with path length is that the HSN was closer to the receivers than to the source. Note that even with a deployment ratio of only 40% of Netlet nodes per stub domain, the average multicast gain was close to 75%. Overall, it can be concluded that ISPs wishing to provide MENU based multicast services can deploy Netlet nodes and attain significant gain over unicast without considering service availability at other parts in the network.

Link Stress:

A common metric by which application level multicast systems distinguish themselves is stress [162]. Stress indicates the number of times that a semantically identical packet traverses a given link. Examining stress will give an estimate on

the level of packet redundancy experienced by network links. Note that, with traditional IP Multicast the stress value never exceeds 1 when all nodes in the network support multicast, i.e. the ideal case.

I quantify the packet redundancy on a per stub domain basis i.e. corresponding to a single ISP. This will allow ISPs to compare the multicast gain against packet redundancy for various percentages of Netlet node deployment. In this analysis, I use a 25 node network which connects 50 receivers to the traffic source. For the purpose of illustration, one of the topologies I use for testing the stress factor (for an average node degree of 4) is shown in Fig. 5.16.

The traffic delivery tree is constructed using the approach described in section 4.3.8. For various levels of Netlet node deployment, the number of duplicate packets traversing each link in the network was recorded. The experiment was repeated for networks with different connectivity levels. Fig. 5.17 shows the change in link stress with increasing number of Netlet nodes in a network. On average, with close to 40% of Netlet nodes, the stress factor reduced by 50% i.e. from 3.5 to 1.72. This is because, as the network connectivity improved there were many optimal shortest path routes available from every node in the network to the HSN which was the virtual source for that network.

Forwarding State Saving with MENU

In MENU, forwarding states are only stored at branch points of the traffic delivery tree. The state saving achieved by not storing at non-branch nodes of the tree, reduces packet processing delays and memory requirements at intermediate network nodes. In this experiment I evaluate the forwarding state saving achieved per established session. The topology used is similar to that as Fig. 5.14. The traffic delivery tree is constructed as described in section 4.3.8. For different levels of user joins to the session, I record the number of routers that had to store packet forwarding state information.

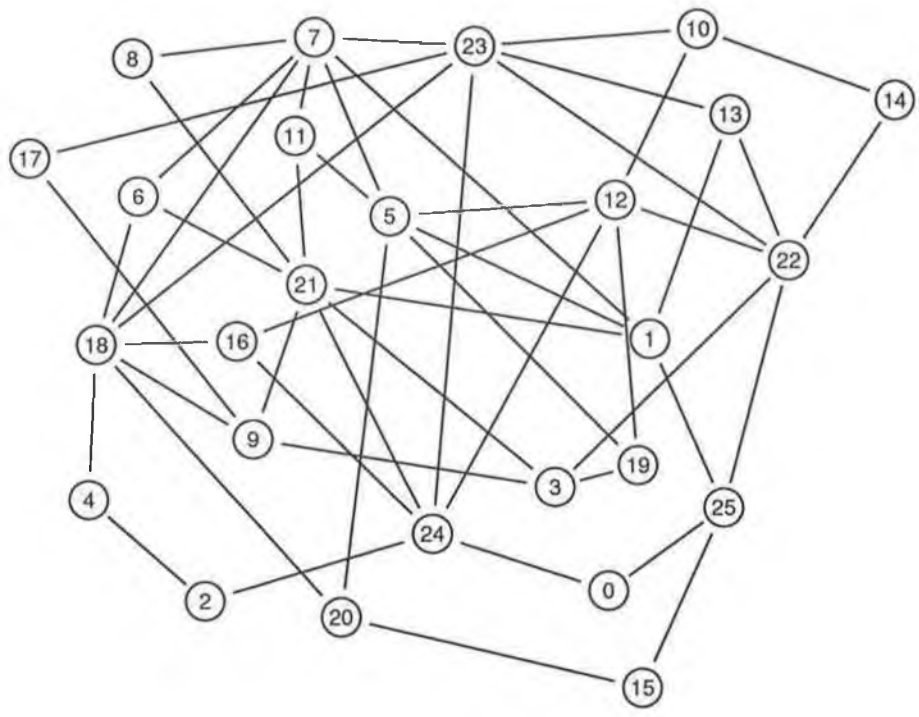


Figure 5.16: Network Topology used for Stress Measurement

Fig. 5.18 shows the state saving achieved by MENU when compared to traditional IP multicast protocols. It is evident that only around 30% of routers present in the delivery tree are required to store session details. This allows a reduction in state storage which inherently reduces packet processing delays, avoids complex packet handling software modules and minimises memory requirements. This reduction improves the scalability of the protocol. Note that, this result is in agreement with the results presented in [148, 150], which reports that close to 70% of the nodes in a tree are non-branch nodes and only function as traffic relay nodes.

Error Recovery Delay

Error recovery delay may be considerably reduced by employing data caching services within the network. Some simulations are presented below which illustrate the improvements in delay which can be achieved. When the data caching

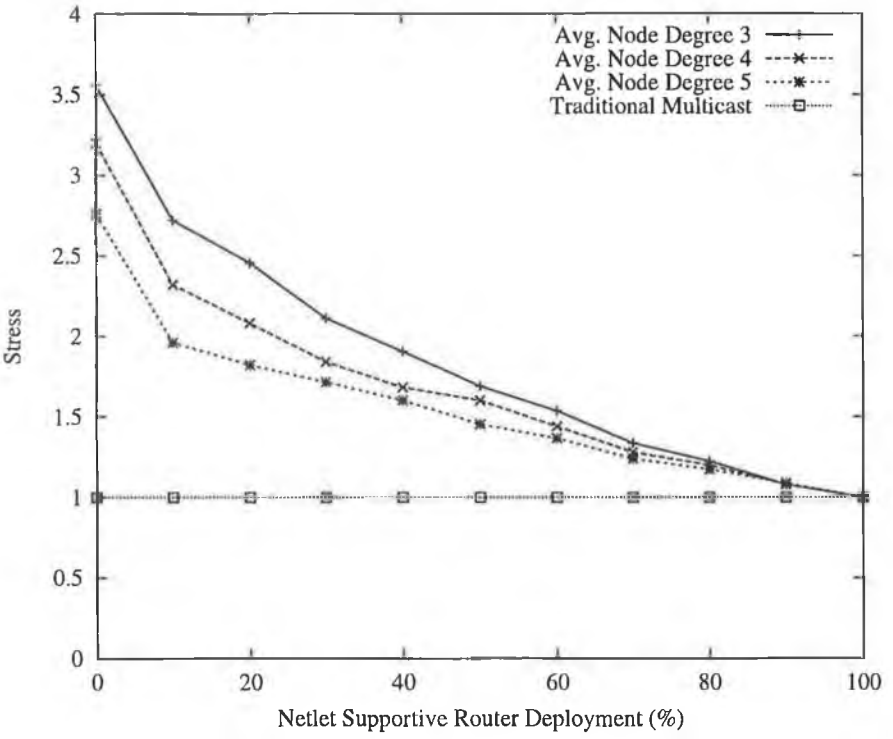


Figure 5.17: Link Stress

service is used, error recovery is initiated by the Hot Spot Delegate (HSD) which (in this simulation) is six hops away from each client node.

The server is positioned variously 6 to 24 hops away from each client (the multicast tree, for simplicity being such that each client is equidistant from the server). Simulations are of the network in Fig. 5.16(with links added as required to balance the multicast tree), where the path bandwidth is set to 10Mb/s, and link delay is set to 20ms.

When data caching is enabled, the error recovery delay is insensitive to the path length from client to server since the effective path length (from client to HSD) is fixed at 6. With data caching disabled the delay increases significantly with path length, as shown in Fig. 5.19.

Additional benefits of HSD based caching are: (i) when an error occurs, the Negative Acknowledge (NACK) packets from clients are contained within the stub domain, thus avoiding the known problem of NACK implosion at the server;

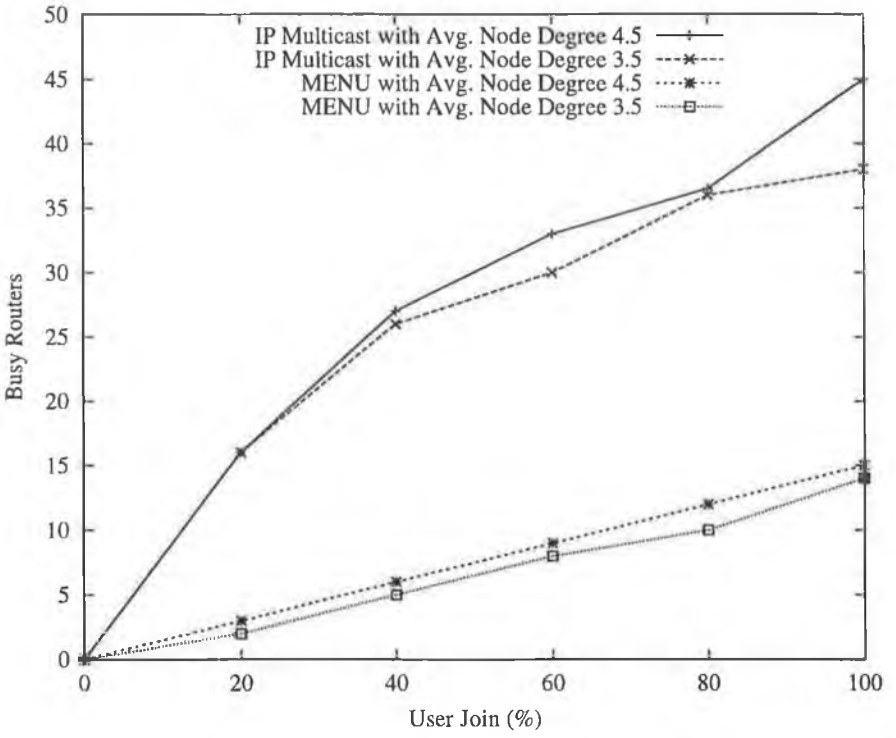


Figure 5.18: Reduction in Forwarding State

and (ii) there is no need for the server to pre-configure error recovery points within the network. Overall, it can be concluded that supporting data caching services significantly reduces error recovery delay.

5.3.4 Remarks

Results show that when only 40% of routers per stub domain can host Netlet and for a mean path length of 23.5, MENU achieves a multicast gain close to 70% and packet redundancy of only 1.72. Furthermore, MENU operates with a 70% state saving, compared to conventional IP multicast protocols thus overcoming their scalability problems. Furthermore, by placing data caching Netlet services, the recovery delay in multicast sessions is also minimised. Overall, MENU provides an efficient means to incrementally build a source customisable secured multicast protocol which is both scalable and reliable.

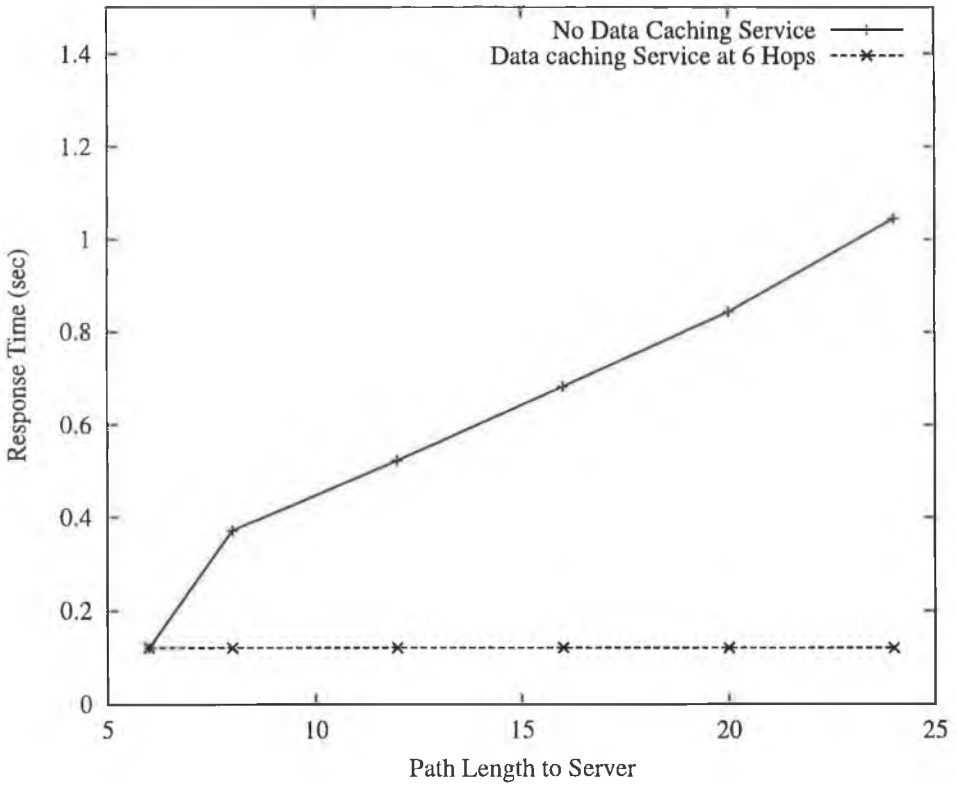


Figure 5.19: Error Recovery Delay for a Multicast Session in MENU

5.4 QoS Support for Network Applications using Netlets

5.4.1 Experiment

In this section I describe the prototype model that I built to demonstrate the mechanism to couple QoS support for end user applications using Netlets (in section 4.4). The aim of this experiment is to confirm the session establishment capability of Netlet services on behalf of end applications. Note this model was not designed to support flow measurement capabilities.

5.4.2 Implementation

The QoS support that is required by end applications operating over a IntServ based stub network will require RSVP signalling. I use the Linux port of the

RSVP[4] package. The RSVP module is programmed in the C language.

I use the Java Native Interface (JNI) to interface Netlets with the native code based RSVP module. I combined the available RSVP APIs under function sets, written in the C langauge to reduce: (a) the number of calls the QoS-support had to place between Java and native domains; and (b) the number of JNI stubs required to interface with the RSVP API.

I developed two distinct C-based program sets for this purpose: the sender and receiver sets, referred to as QoS-sender and QoS-receiver respectively. The sender set encapsulates the QoS parameter initialisation (based on parameters received from the Netlet), socket creation for communication and RSVP session start-up methods which also includes reservation checks and confirmations. The receiver set contains APIs, which allows to receive reservation requests, sending reservations messages and their corresponding error and confirmation messages.

The experimental setup that I use for this test is shown in Fig. 5.20. I perform tests to confirm the connectivity established between Netlets and the traffic source node. The test environment has two Netlet nodes serving as edges of a stub network and a manager Netlet node to process and coordinate QoS support requests. For the purpose of testing I use two end applications working in traffic receiving mode (on Traffic Receiver as in Fig. 5.20). Hence, it has to send the RESV message [2] towards the traffic source. The end nodes hosting the application are configured to start the communication session through the two edge nodes of the stub network. A web utility (Fig. 4.12) is used to receive QoS support requests from users.

I use the RSVPSignalling Netlet for this test. The role of this Netlet is to migrate to the two edge nodes (E1 and E2 as in Fig. 5.20) and install the Java native service, which is responsible for triggering the C based QoS-sender program. Note the JNI class for triggering is encapsulated within the Netlet and also in-

[4]RSVP Port Under Linux, http://www.isi.edu/div7/rsvp/release.html

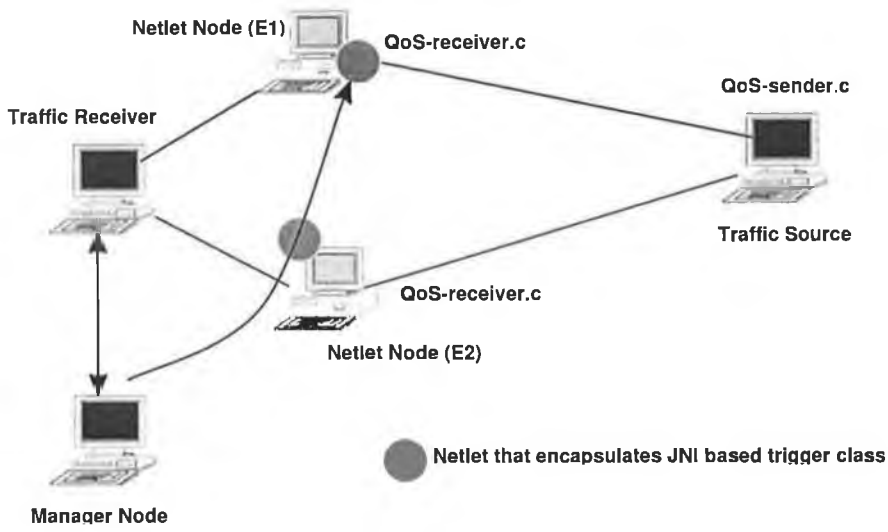


Figure 5.20: Experimental Setup

cluded the address of the FilterSpec object of the traffic receiver initiating reservation.

On test startup, requests are generated by client node to the manager node. On receiving the requests, the manager node then launches a signalling Netlet, referred to as RSVPSignallingNetlet. This Netlet then migrates to the edge nodes and installs the QoS support service for initiating reservations. The QoS-receiver program are configured to run as a background process at the traffic source (as in Fig. 5.20).

Following this, on identifying the occurrence of a flow matching the FilterSpec parameter the RSVPSignallingNetlet initiated RSVP signalling messages using the QoS-sender program. The receiver node replied with acceptance messages for the reservation requests, thus confirming the validity of the session initiated by the RSVPSignallingNetlet Netlet. This implementation running on a small network, proves the validity of the concept.

5.4.3 Deployment in Large Networks

Practical issues need to be addressed before this application can be deployed on a large scale. These issues include the following:

Multiple Manger Nodes: Depending on the size of the stub network, multiple manager Netlet nodes for QoS support can be used to process end user requests. This will prevent user requests from overloading a single node, thus avoiding a single point of failure in the network.

Non-Active Stub Network Edges: When non-active segments are present along the stub network edge, Gateway Netlet Nodes (GNN) can be used to enable signalling support to users attached to this segment of the network. In this case, the web-based utility can be enhanced to provide the user with a list of GNNs through which the end users can choose to route their traffic into the QoS aware network. Based on the GNN selected by the end user, Netlets can be deployed by the manager node to invoke signalling support for end applications.

Wireless Subnets: Large scale deployment will require the Netlets architecture to function in wireless subnets. Wireless networks have unique QoS requirements. The Netlets architecture is well suited to supporting QoS in such networks, since it provides a method to offer tailored network services at the interface between the wired and wireless segments of the network.

5.4.4 Remarks

I described the implementation of a mechanism to integrate QoS support to legacy network application using Netlets. A proof of concept implementation has been deployed in a laboratory environment. This approach can be used as a permanent long-term solution to interface network applications with emerging QoS models on demand. Adapting this approach insulates future network applications required to be aware of the specifics of the network support for QoS.

5.5 General Performance Characteristics of a Netlet Node

The performance of Netlet nodes[5], in terms of throughput and latency, will determine their suitability to host network services in commercial network environments. Ideally, a Netlet node should be able to provide the base case functionality of a typical IP router, i.e. packet forwarding based on the destination address.

Packet Handling Mechanism: The Netlet packet handling mechanism differs for unicast and multicast based services.

- **Forwarding Netlet for Unicast Communication:** In this case, the forwarding Netlet service receives packets from the Packet Communication Engine[6], and then forwards them towards their destination addresses based on the routing table entries. We refer to this packet forwarding Netlet as an **U-Netlet** in the following discussion.

- **Forwarding Netlet for Multicast Communication:** In this case, the forwarding Netlet service receives packets directly (i.e. on the specific port number registered with the JVM), It then performs an address translation to denote the next hop node to which the packet should be sent and forwards the packets towards this address. We refer to this packet forwarding Netlet as an **M-Netlet** in the following discussion.

5.5.1 Performance Evaluation for U-Netlet

The experimental setup I use for performing these measurements is shown in Fig. 5.21. It consists of a Data Quality Analyser (DQA[7]) [165], and a Pentium-based PC (with 256KBytes RAM and 850MHz clock speed and with RedHat Linux dis-

[5]By characteristics of a Netlet node, I refer to the NRE layer of the network node.
[6]Recall that this module is responsible for packet capture and classification
[7]MD1230A Data Quality Analyser is a portable measuring instrument to evaluate performance characteristics of IP network elements.

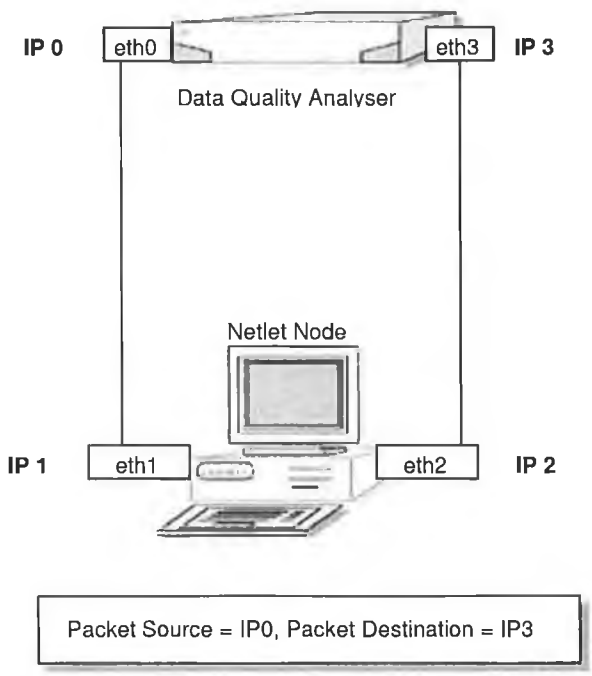


Figure 5.21: Experimental Setup for Measuring Performance in Unicast Communication

tribution, 7.1). The PC hosts the Netlet Runtime Environment, thus acting as a Netlet node. The DQA and the Netlet node use $100Mbps$ Ethernet network interfaces and are bridged using crossover cables. Furthermore, the Netlet node is configured to act as a router gateway for the DQA.

The DQA is programmed to send and receive UDP packets through the Netlet Node (see Fig. 5.21). Furthermore, the UDP packets are made to include a reference to the U-Netlet service. The data forwarding service is loaded and activated at the Netlet node prior to the flow of data. This is to avoid delays caused due to service discovery and deployment on incoming packets.

Packet Forwarding Latency Across a U-Netlet Service

I first measured the latency introduced when forwarding packets using the U-Netlet service across the Netlet node. To place the performance of this measurement in context, I also perform measurements to analyse the latency experienced

when forwarding packets: (i) at the kernel level, using a PC running the Linux OS; and (ii) at the user level, using a C-based implementation of the forwarding service, referred to as C-forward.

The packet sizes used for testing are in the range of 64Bytes to 1500Bytes in length. For each test configuration and packet size, $5,000$ packets are sent and received through the Netlet node by the DQA. The DQA is configured to generate packets once every $500ms$. This interval is sufficient to ensure that forwarding services have sufficient time to process the packets. The experiments are repeated three times each, and the median of the three trials are taken. The latency to cross the Netlet node is measured using tcpdump. A modified Linux kernel is used at the Netlet node to accurately timestamp Ethernet frames, with an error less than 800ns.

Fig. 5.22 shows the results of these measurements. In all cases, the packet forwarding delay increases with packet size. As expected, forwarding packets at the kernel level results in minimal delays (ranging from 28 to 308 μseconds). The delay for the C-forward is only 40μseconds more than for forwarding at the kernel level. In the case of the U-Netlet, the additional delay is close to 250μseconds regardless of the packet size.

I analysed the processing overhead incurred by the user level forwarding services, both U-Netlet and C services, relative to that of kernel level forwarding. Processing overhead was calculated using the following formula (A.10). Fig. 5.23 shows the results of this analysis.

$$Relative \quad Processing - Overhead \quad = \frac{T_{Userspace} - T_{Kernel}}{T_{Kernel}} \tag{5.6}$$

where $T_{Userspace}$ denotes the forwarding latency when employing a user level service, while T_{Kernel} denotes the delay incurred at the kernel level.

The packet forwarding latency across the Netlet node, i.e. using a U-Netlet, is insignificant (Fig. 5.22). This suggests that Netlet nodes can be employed for

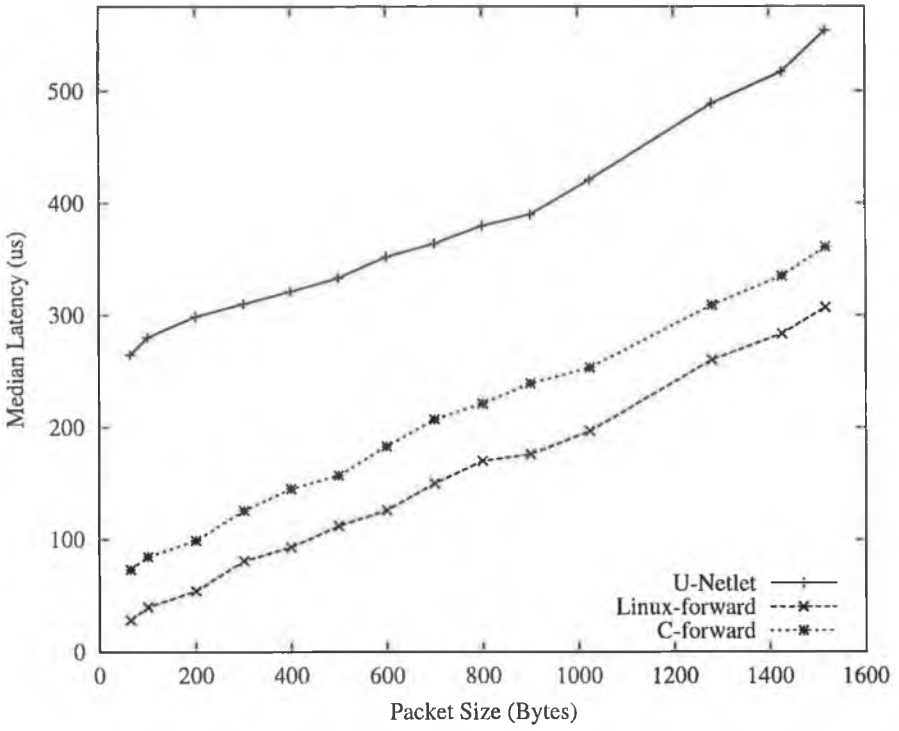


Figure 5.22: Forwarding Latency Across a U-Netlet Service

delay sensitive real time applications such as media streaming. The excess delay incurred by the Netlet service in comparison to other forwarding services may be attributed to the Java Virtual Machine (JVM), since each instruction has to be mapped (i.e. loaded, decoded, and invoked) by the interpreter before execution. Interpretation times of byte-codes are normally ten times more than execution of native machine code [83].

The results in Fig. 5.23 shows that the relative cost of using a Netlet node for packet forwarding decreases as packet size increases. This is because the delay incurred to process packets at the kernel level increases with increasing packet size, while the Netlet processing delay remains relatively constant.

Throughput of a Netlet Node when using a U-Netlet Service

To study the throughput characteristics of the Netlet node for supporting Class-U services, the input load from the DQA is gradually increased to determine the

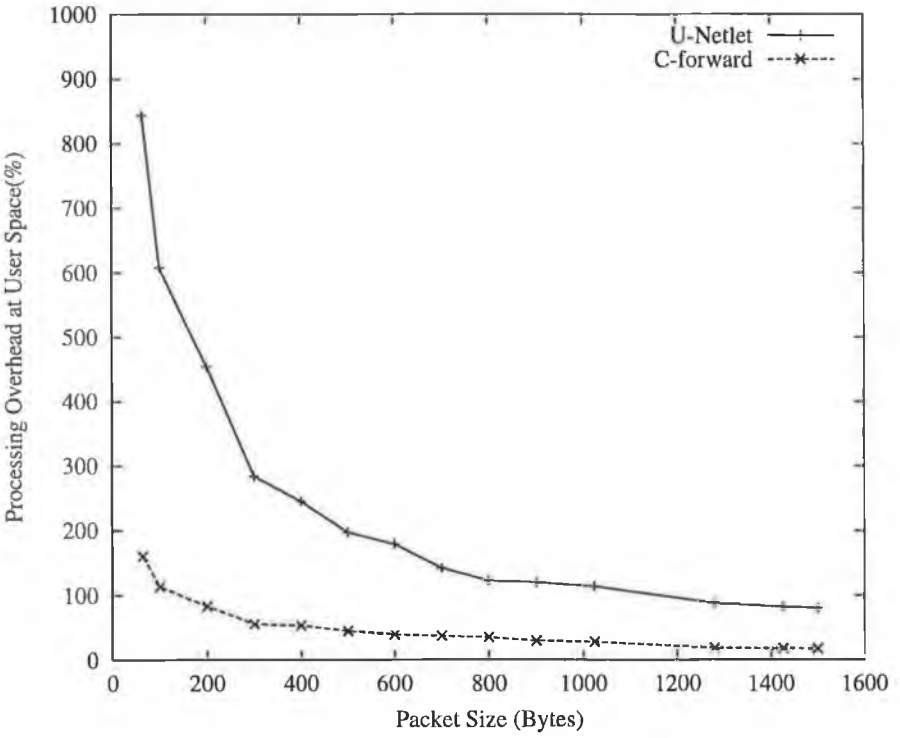


Figure 5.23: Processing Overhead in C-forward and U-Netlet Service Relative to Kernel Level Forwarding

maximum effective throughput which the Netlet service can support. Fig. 5.24 shows the effective throughput of the Netlet service for different packet sizes (64Bytes to 1500Bytes). The Netlet service achieves a maximum effective output rate close to 35Mb/s for an input rate of 100Mb/s and packet size of 1500 Bytes.

When using the same hardware platform as a PC based router running Linux and performing packet forwarding at the kernel level, wire-speed operation is achieved for an input rate of 100Mb/s and packet size of 1500 Bytes. The C-forward service achieves an effective throughput of 78Mb/s for an input rate of 100Mb/s and packet size of 1500Bytes, as shown in Fig. 5.25.

The decrease in throughput of the Netlet node is primarily due to two reasons. Firstly, packet capturing is performed by a Java Native Interface (JNI) code which is then responsible for handing over packets to the U-Netlet service. This incurrs additional delays (i.e. system calls, data copy overheads across the kernel and

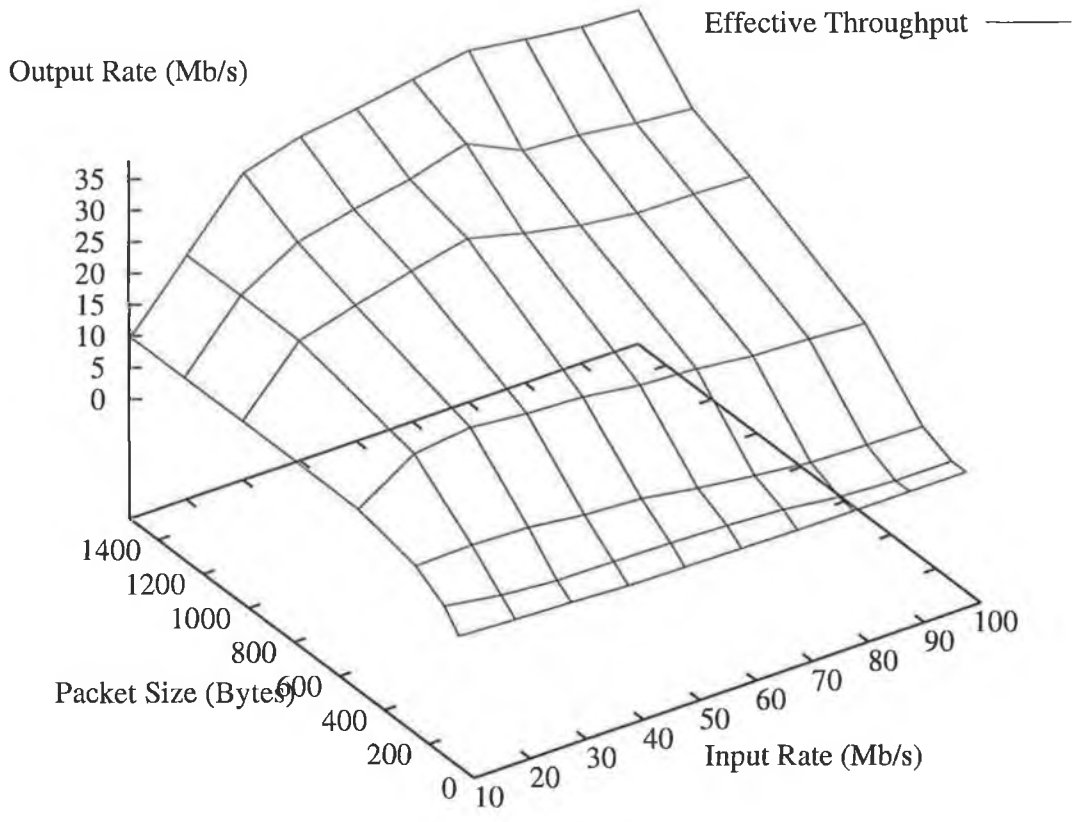


Figure 5.24: Throughput of a U-Netlet Service

user space boundaries) causing a decrease in throughput.

Secondly, the JVM is burdened with frequent context switch operations. I observed that with the increase in packet size the throughput of Java based service increases significantly; this again proves that, with minimal context switches within the JVM, throughput increases. The throughput ratio can be improved by employing hardware based bytecode interpreters (e.g. [166, 167]) and Java based processors [118] which produce an increase in speed of 5 to 10 times over software implemented JVMs. It may be expected that the performance achievable in executing bytecode will improve significantly in the near future using such techniques. Some acceleration techniques peculiar to the Netlet node problem may

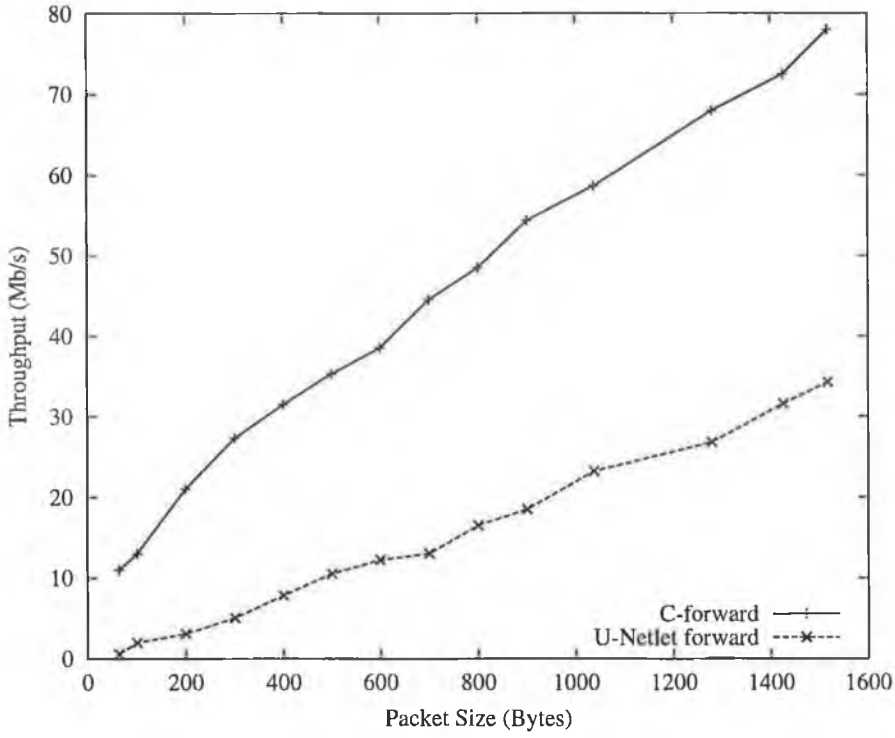


Figure 5.25: Throughput of the Netlet Forwarding Service and C-forward

also be applied. These are:

- Active Packet Classifiers - hardware based packet classifiers can be employed at network nodes to improve the throughput rate. Recall that packets requiring active processing are differentiated from regular datagram traffic using special packet options, such as the router alert field in IP packets or using special purpose labels in MPLS networks. Hardware based packet classifiers at network nodes can be configured to deliver such packets at wire-speed to the NRE for special processing, thus avoiding the delays incurred by performing packet capture and handover using software modules;

- Compilation to Native Code - performing code interpretation instead of code compilation slows downs processing considerably. A solution to enhance performance is to compile frequently used network services to binary

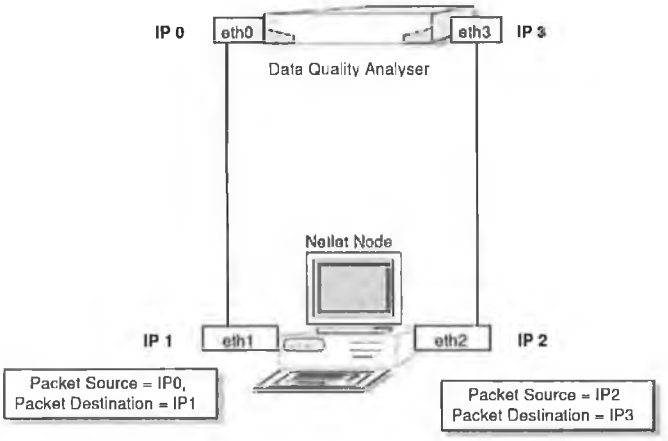


Figure 5.26: Experimental Setup for M-Netlet Service

at the local node. Cache maintenance and validation policies can be used to control the service population and overload conditions at the node.

5.5.2 Performance Evaluation for M-Netlet

Here, I measure the performance characteristics of a M-Netlet operating service. The M-Netlet operating at the Netlet node (Fig. 5.26), receives incoming packets from the DQA at a pre-configured port, changes its destination address according to the entries in the data distribution table (e.g. in Fig. 5.26, the address is changed from IP1 to IP3), and forwards them towards their destination address.

Packet Forwarding Latency Across a M-Netlet Service

The procedure to measure the packet forwarding latency is similar to that used for U-Netlet type service (see section 5.5.1). Fig. 5.27 shows the results obtained. The packet forwarding latency for the M-Netlet service is around 15μseconds less than for the U-Netlet service, a marginal improvement.

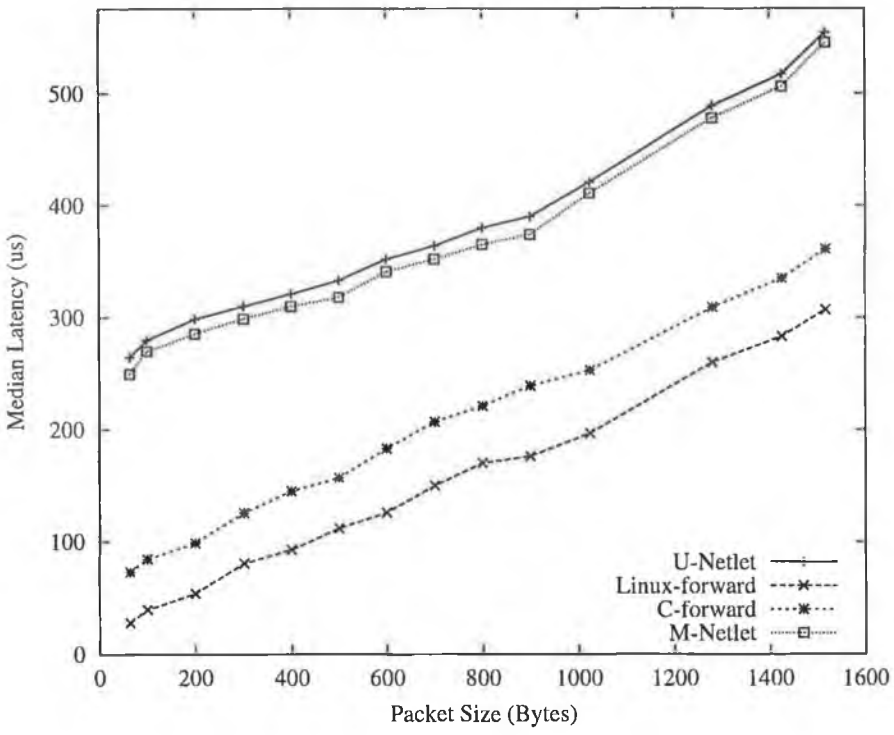


Figure 5.27: Forwarding Latency Across a M-Netlet Service

Throughput of a Netlet Node when using a M-Netlet Service

Next, I measure the effective throughput that a M-Netlet service can achieve. The procedure for performing the measurement is similar to that adopted for U-Netlet service. Fig. 5.28 shows the results. The M-Netlet service achieves a maximum effective output rate close to 60Mb/s for an input rate of 100Mb/s and packet size of 1500 Bytes.

This is considerably better than the figure for the U-Netlet service which was only 35Mb/s. The underlying reason for the increase in throughput is that – packets in the M-Netlet are not processed through the Packet Communication Engine, but rather are handed directly over to the M-Netlet service by the JVM, based on the port number registered by M-Netlet with the JVM. This feature minimises the frequent context switches within the JVM and packet handover delays. An increase in packet size increases the throughput of the service significantly; this is

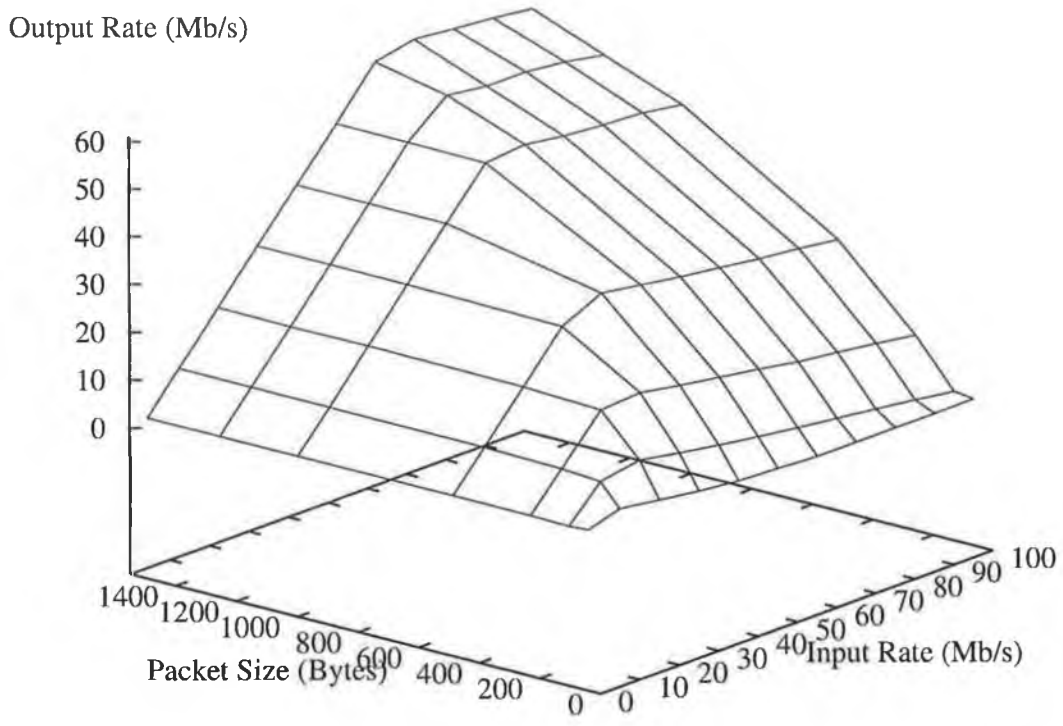


Figure 5.28: Throughput of a M-Netlet Service

again because, with minimal context switch operations within the JVM, throughput increases.

5.5.3 Unicast-Netlet and Multicast-Netlet Services

Benefits of M-Netlet Service: The fundamental feature of the M-Netlet service is that the destination address of the packet is mapped to a logical group of destinations in the network. Such a service is applicable to a wide variety of existing or proposed network services such as anycast [11], concast [168] and pamcast [169]. The Replication Netlet service in the MENU protocol belongs to this class.

Applications of U-Netlet Service: Existing active network systems forward pack-

ets using the technique employed by U-Netlets. For example, an active node featuring mobile code and Java Development Kit (JDK) v1.2, was reported to achieve an effective throughput close to 33Mb/s [170], which is in agreement with our results obtained for the U-Netlet service. Although the performance of U-Netlets is inferior to that of M-Netlets, they are appropriate for applications, which require selective packet processing, such as the solutions to the problem of reservation gaps and server selection. In the case of RSVP gaps, the gap manager Netlets at QoS-nodes are only required to process the RSVP messages pertaining to a flow rather than all its datagrams. Thus the limited throughput of the U-Netlet is of less concern.

5.6 Dynamic Deployment of Netlet Services at Runtime

The applications presented in Chapter 4 rely on dynamic deployment of Netlet services for service provisioning. When an incoming packet requests a service which is not present locally, the local node immediately invokes the service discovery protocol. The delay to dynamically discover and deploy Netlets can affect the overall quality of service perceived by end applications. Here, I perform measurements to estimate the delay involved in setting up services dynamically at a Netlet node. This would give an estimate of the packet waiting time at a node. The analysis is performed using live measurements on a testbed constructed in a laboratory environment.

The experimental setup for this set of analysis is shown in Fig. 5.29. The DQA is programmed to generate packets of size 512 Bytes with a reference to the service that should process them, in this case the U-Netlet forwarding service. The Netlet node functions as an IP router. Two PCs are configured to act as code servers and are loaded with the bytecode file of the U-Netlet forwarding service.

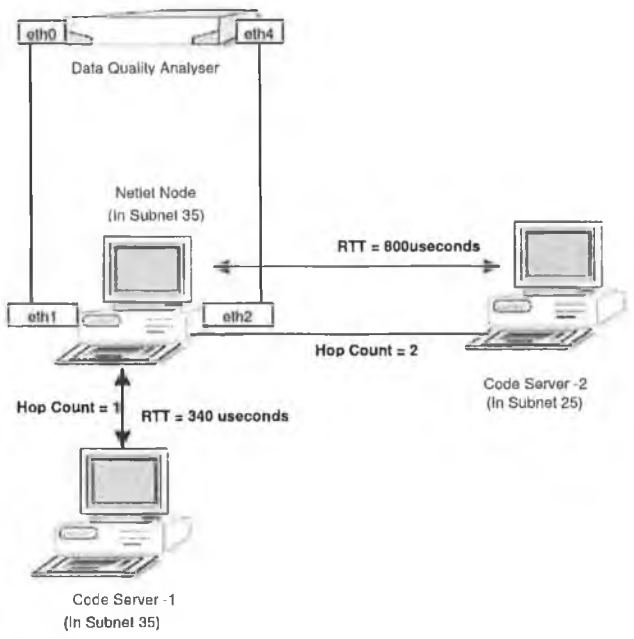


Figure 5.29: Experiment Setup To Evaluate the Reactive Service Deployment Scheme

The following tests are performed to study the packet waiting at a node:

- **Test1:** Forwarding service is loaded prior to the arrival of packets. However, the service is only activated by the arrival of a packet. After transmitting each packet, the service thread corresponding to the data forwarding Netlet is terminated by the Netlet Management Engine. This is to ensure that the service must be reactivated when the next packet arrives.

- **Test2:** On arrival of a packet requesting the service, the Netlet node downloads the bytecode file of the U-Netlet service from the server. Here a server that is a single hop away is used by the Netlet node. After transmitting each packet, the service is removed from the local store so that the next packet will also trigger the loading of code.

- **Test3:** The above test is repeated with the server two hops away.

The results obtained are summarised in Table 5.1. It can be concluded that caching services at Netlet nodes is highly beneficial for maintaining stable per-

Code Distribution Method	Setup Time	Total Forwarding Latency
Local Cache (Service Active)	0ms	333 μs
Local Cache (Service Inactive)	0ms	890 μs
Code Server-1 (1 hop away)	225 ms	225.9 ms
Code Server-2 (2 hops away)	267 ms	267.9 ms

Table 5.1: Forwarding latency for a packet of 512 Bytes in length when supporting dynamic deployment

formance levels within the network. As expected, loading from the closer server caused a lower waiting time for packets. The Stigmergy protocol (see section 3.3.2), finds the closest node that hosts the required service (e.g. code server-1 in Fig. 5.29). Such a scheme minimises the service discovery time, thereby improving the end-to-end service levels perceived by applications.

5.7 Service Deployment Latencies

5.7.1 Reactive Service Deployment

Service Deployment Latency in Wide Area Networks

I use simulation to study the service deployment latencies in wide area networks, such as the Internet. The service deployment delay consists of: (i) the delay to obtain the necessary bytecode of the service; and (ii) the delay to instantiate the service locally. The time to setup a service locally was found to be around $557\mu seconds$ for the data forwarding Netlet service (Table 5.1) and thus is insignificant. Hence, the time to download the required bytecodes constitutes the significant element of the total service deployment time.

I use the Network Simulator (ns) package [171] for performing the simulations. Our analysis was performed using the core-stub network topologies generated by the GT-ITM [163] package. The topology used in our simulations is shown in Fig. 5.30. The network has an average node degree 3.6, with effective path bandwidth of 10Mbps and link delay of 30ms.

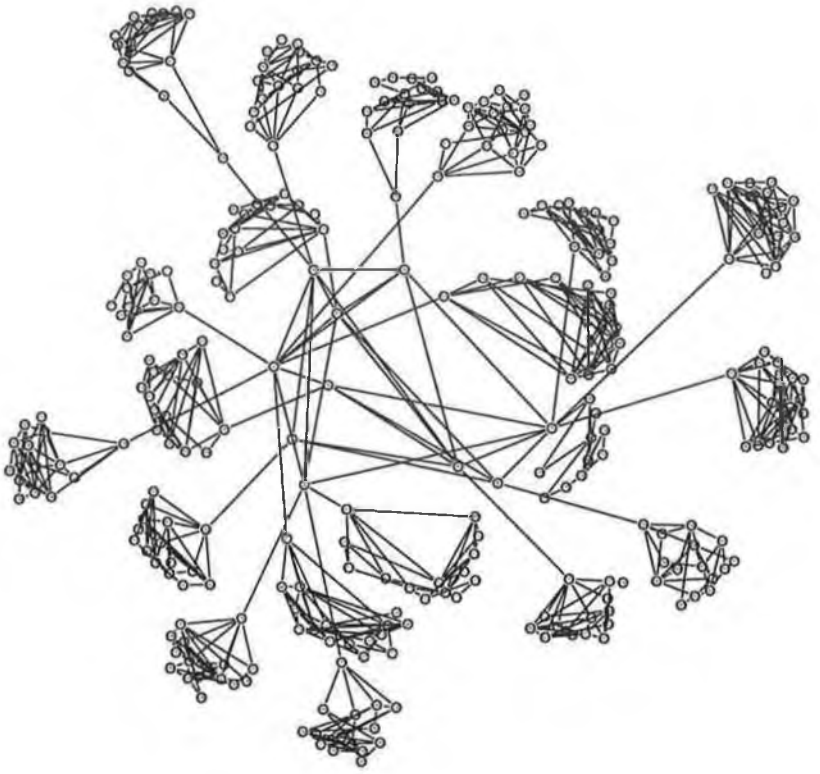


Figure 5.30: Network Topology used for Analysing Stigmergy

In the simulation setup, three nodes are selected to represent the Netlet node requesting service (client), the node containing the required service (cache) and the home node of the Netlet. The distance between the client and the home node is configured to be 14 hops. The location of the cache is selected at a midway point between the client and the home node, i.e. 7 hops in this case. The server and cache nodes are configured to host various services of sizes in the range 500Bytes to 32KBytes. TCP is used for communication purposes.

In the first set of experiments, we measure the service deployment delay without caching. The client node is configured to contact the home node for the byte-code file necessary to deploy a service. Fig. 5.31 shows the results of this analysis.

In the next set of experiments, the client uses the Stigmergy protocol to discover the cache node, and requests the code from it. Fig. 5.31 shows the results of this analysis. Note this delay is sum of the discovery and code fetching delays.

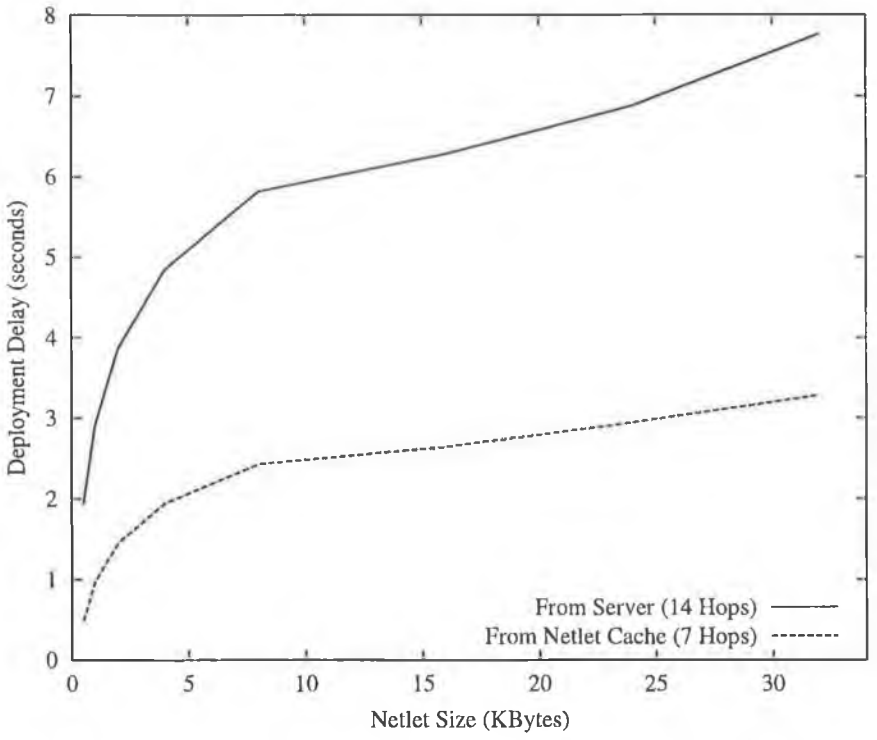


Figure 5.31: Service Deployment Delay in the Reactive Model

Performance of the Stigmergy Protocol

I compared the performance of the Stigmergy protocol to that without caching. I measured the variation of downloading delay incurred by the Stigmergy protocol as a function of cache distance from the client node; and then, using equation 5.7 I calculated the normalised delay of the cache-less approach.

$$Normalised \quad Overhead \quad of \quad Cache-less \quad = \frac{T_{Standard} - T_{Cache}}{T_{Cache}} \qquad (5.7)$$

where, $T_{Standard}$ denotes the time to obtain the service from the home node, while T_{cache} denotes the time to download from the cache.

The simulation setup for the measurements involved a server (the home node), client and a cache node. The server was configured to be 17 hops away from the client. For every measurement I varied the cache location in relation to the client from a distance of 2 to 14 hop counts, in increments of 2. I also measured the

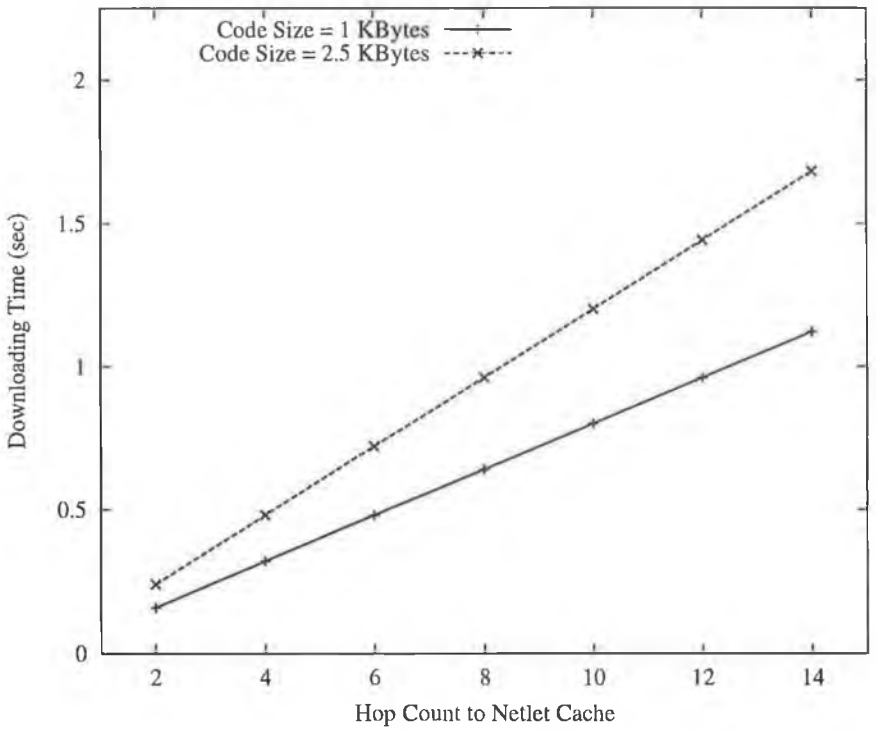


Figure 5.32: Service Deployment Time for various Cache Locations within the Network

downloading delays for various client locations and server positions. The measurements were performed for code sizes of 500Bytes and 2.5KBytes. Fig. 5.32 shows the results from this analysis.

Next, using equation 5.7, I evaluated the overhead incurred by the cache-less approach relative to the Stigmergy protocol. The results are presented in Fig. 5.33.

Discussion

Fig. 5.31 shows the variation in downloading delay as a function of code size. By exploiting the caches present at Netlet nodes, the Stigmergy protocol achieves a near three fold reduction in downloading delay. The cache-less approach introduces considerably more delay than the Stigmergy protocol. This is shown in Fig. 5.33, where the delay associated with the cache-less approach is shown

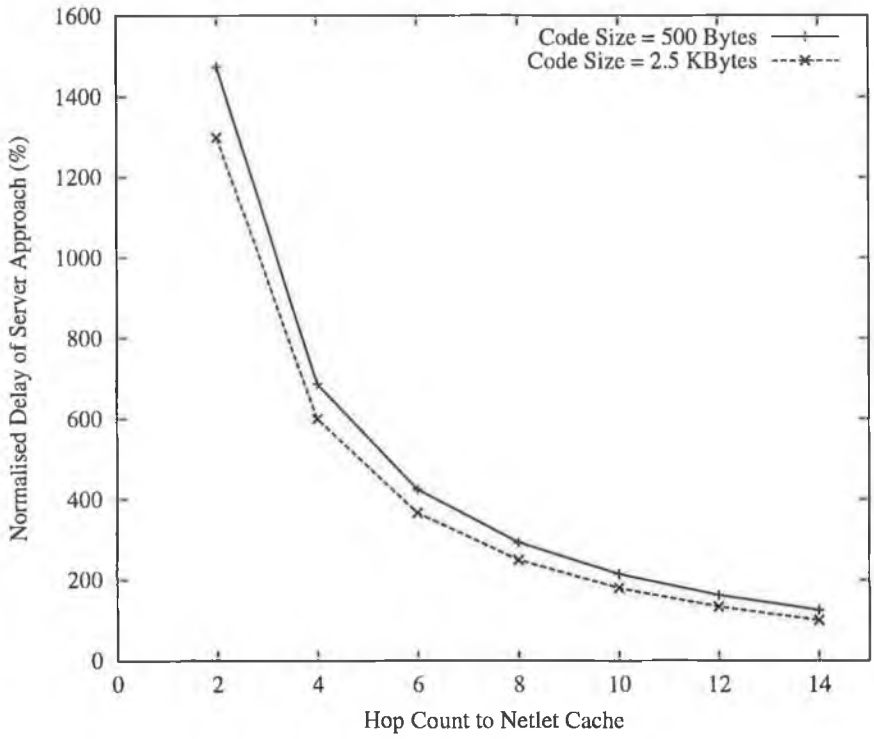


Figure 5.33: Normalised Delay of the Cache-less Approach for Service Deployment

as a proportion of the corresponding delay for a network using the Stigmergy protocol.

Excess delays imposed by active network systems on packets requesting services will adversely affect end applications. Furthermore, in such systems, packets will be dropped if the required set of services are not deployed within a predefined time bound. This will in turn increase the number of failed connections within the network. Hence, the complexity of the Stigmergy protocol, in comparison to the cache-less scheme, is justified by the superior delay performance of the network when it is used.

5.7.2 Proactive Service Deployment

Methods presented in Chapter 4 to support multicasting and server selection illustrates the potential of user representative services within the network. In such

protocols, a server, on identifying a community of interest at a specific location in the Internet dynamically deploys the required set of services to provide packet processing support.

I performed simple experiments to gauge the time scales over which active nodes can be discovered in the Internet. This would allow us to assess the service deployment delays that could be of expected when operating in wide area network environments.

The performance of a proactive service deployment scheme (section 3.3.1) depends on two major factors. They are: (i) the time taken to discover active nodes at a specific location in the network; and (ii) the time required to transfer services to those locations.

Delay to Discover Netlet Nodes in the Internet

The DNS-based scheme presented in section 3.3.1 can be used to discover Netlet nodes in the Internet. In this scheme, the discovery delay consists of: (i) the delay to obtain the node lists of each domain at which Netlet nodes are required to be discovered; and (ii) the time taken to exchange confirmation messages, e.g. "hello", in order to confirm that the discovered nodes support Netlet execution.

For this purpose, I randomly selected five domains in the Internet that were located at various geographical locations (5.34).

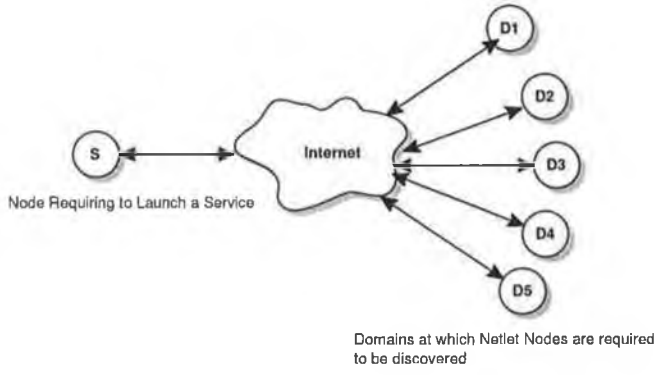


Figure 5.34: Network Topology used for Service Deployment

I used the dig[8] DNS tool to measure the time required to obtain the node list from each domain. These domains of course did not host any active nodes. However, the delay to obtain the node list is indicative of the actual delay that will be experienced when performing active node discovery over the Internet. The delay to exchange confirmation messages between the node requiring to launch a service, S in Fig. 5.34, and the discovered network nodes is equivalent to that of the Round Trip Time (RTT) between them.

Country	Discovery Delay (ms)	Round Trip Time (ms)	Hops
Ireland	18	1.91	6
South Africa	259	196	22
USA	276	108.45	13
Brazil	491	236.36	16
Australia	856	335.48	25

Table 5.2: Delay to Discover Active Nodes at various Geographical Locations

The total service deployment time when using the current Netlet implementation can be calculated as follows. Recall that I use TCP for transferring Netlet services within the network. For the example, assuming a slow-start TCP[9], I can calculate the deployment time using equation 5.8 [172]. Here, N represents the size of the Netlet to be transferred.

$$Total \quad Transfer \quad Time = RTT * (\lceil log_2 \rceil) N + 1) \tag{5.8}$$

Table 5.2 shows the results of the experiment. From Table 5.2 it can be concluded that it is possible to discover active nodes in the Internet with acceptable delays. This confirms the viability of proactive service deployment scheme with Netlets across a wide area network.

[8]http://www.dns.net/dnsrd/tools.html

[9]Slow-start TCP doubles its window every RTT before congestion is detected.

5.8 Issues in Netlet Deployment

Netlet Location: The high volume of traffic carried by the Internet core prohibits extensive packet processing, and this trend is expected to continue. Hence, any active networking system which requires packet processing in the core is likely to prove impractical. This clearly demonstrates that edge network domains (in which routing nodes have more free memory resources and CPU cycles) are where packet processing support can be provided.

However, such a limitation on the location of active nodes does not seriously limit the usefulness of the Netlets architecture. It may be expected in many applications that the best performance will be achieved by locating Netlet services close to the end-users. Since the end-users by definition lie at the edges of the network, it follows that, given a free choice of Netlet location, Netlet services will nonetheless be clustered near the edges of the network.

Thus, although prohibiting Netlets from the network core may result in their suboptimal deployment, the resulting performance degradation may be minor. For example, the MENU protocol exploits the existence of communities of interest in the Internet to efficiently offer multicast service using Netlets, the resulting multicast trees approaching the efficiency of an optimal solution.

Netlet Performance: The packet forwarding latency across the Netlet Runtime Environment is insignificant (i.e. adding just 250μseconds more delay than traditional forwarding in the prototype implementation, as shown in Fig. 5.22). This suggests that Netlet nodes can be employed for delay sensitive real time applications such as media streaming.

The throughput characteristics of the Netlet node vary for unicast and multicast (i.e. using MENU) communications. In the former case, the prototype was only able to achieve around 35Mb/s (see Fig. 5.24), while in the latter case, the prototype achieved an effective throughput of around 60Mb/s (see Fig. 5.28).

The key reason for this increase is due to the fact that MENU is designed in such a way to avoid packet capture and classification delays incurred at the Packet Communication Engine of a Netlet node. Since in general it may be expected that not every packet will require processing by the Netlet Runtime Environment, this level of throughput will be sufficient for many applications. Custom hardware would be required to get Netlet throughput close to (say) 1Gb/s. This is a topic for further study.

Netlet Deployment Levels: How many Netlet nodes will have to be present in a network in order to exhibit good performance levels for end applications?

There is no simple answer to this question. However, an indication of the levels required may be found by looking at some demanding applications and studying how their performance is affected by the level of Netlet deployment. Two of the most demanding applications are QoS and multicast. We found that with only around 40% of routers being active per stub domain, the Netlet based solutions to QoS and multicast (sections 5.1 and 5.3) exhibit good performance levels.

5.9 Summary

In this chapter, I presented results from experiments that were carried out to evaluate the set of applications described in Chapter 4. Furthermore, generic tests to evaluate the Netlet prototype model were presented.

I first presented experiments to study the problem of reservation gaps. I showed that robust end-to-end QoS support can be provided using gap managing Netlets even when a significant number of network nodes (40%) do not support QoS, especially when employing the *S-MR* and *M-RS* routing algorithms.

Next, I presented an evaluation of the server selection approach using Netlet services. The director service was able to capture and route requests to the

best performing server (as identified using measurement probes), thus achieving load distribution across the servers and minimising the client-perceived response time significantly. By locating Netlets close to client nodes, the number of measurement probes required was reduced, thus realising a scalable model.

The performance of the MENU protocol was also evaluated. The results obtained show that when only 40% of routers per stub domain can host Netlet, MENU achieves a multicast gain close to 70% and a packet redundancy level of only 1.72. Furthermore, MENU requires only 30% of the amount of state information used by conventional IP multicast protocols, thus overcoming their scalability problems. MENU provides an efficient means to incrementally build a source customisable secured multicast protocol which is both scalable and reliable.

I then described the implementation of a mechanism to integrate QoS support to legacy network application using Netlets. This mechanism insulates both future and legacy network applications from the requirement to be aware of the specifics of the network support for QoS.

The remainder of this chapter presented generic tests to evaluate the performance characteristics of the Netlet prototype. Experimental results show that the Netlet prototype provides low packet processing latencies for both multicast and unicast communication. When implemented on a Linux-based soft router using 100Mb/s interfaces, it exhibits a high throughput ratio for multicast communication (of around 60Mb/s), although throughput is reduced for unicast communication (to around 35Mb/s). The MENU based multicast mechanism proposed is applicable to a wide variety of existing or proposed network services such as anycast [11], concast [168] and pamcast [169]. The unicast based communication model is appropriate for applications which require selective packet processing, such as the solutions to the problems of reservation gaps and server selection.

Finally, I evaluated the service deployment mechanisms used to distribute Netlet services in the network. By exploiting the caches present at Netlet nodes,

the Stigmergy protocol achieves a near three fold reduction in Netlet downloading delay. Furthermore, experiments conducted to evaluate the specifics of the proactive deployment scheme used by Netlets, confirms its viability of operating in a wide area network.

Overall, the proposed architecture shows promise as an infrastructure for future Netlet service development and deployment.

Chapter 6

Conclusions

6.1 Contributions

I have endeavoured in this thesis to demonstrate that network programmability can be introduced into existing networks using Netlets, in such a way that they can be used to augment and support existing network protocols. I have also implemented a number of new network services using Netlets.

6.1.1 Netlets Network

Integrating Netlet Runtime Environment into IP Networks: I presented the design of the Netlet Runtime Environment which supports execution of Netlet services at network nodes. The two layer model of the Netlet node performs forwarding operations of regular packets using regular IP network elements, while packet processing is performed using the software-based NRE layer. This design feature allows programmability to be integrated as a "value-added" service to existing networks, while still being able to maintain packet forwarding levels for "regular" packets comparable to that of the current networking environments.

Proactive Service Deployment: I described the use of Netlet services to represent users (e.g. Hot Spot Delegates in the MENU protocol, as in section 4.3)

at distributed points in the network. In this case, Netlets were required to migrate under their own control and to deploy new services at various points in the network, in order to provide packet processing support on behalf of end users. For this purpose, Netlet services would have to discover Netlet nodes to host the service in various regions (e.g. different domains).

I proposed a DNS-based discovery scheme to locate active nodes in the Internet. This approach leverages an existing Internet protocol. The scheme is distributed and thus features scalability. Experimental results showed that it is possible to rapidly locate active nodes in a wide area network using this protocol.

Reactive Service Deployment: I proposed a service discovery protocol, referred to as Stigmergy, which supports the discovery of Netlet services in the network. The key feature of the Stigmergy protocol is that each Autonomous System in the network is treated as an independent two level caching structure in which the upper level, $L1$, contains pointers to Netlet services that are present in the lower level, $L0$. This protocol, by self-organising network nodes that are under a common administrative control into virtual cache clusters, maximises the chances of discovering the required services within minimal latencies. The Stigmergy protocol is completely distributed and follows a best-effort cache co-operation model. Furthermore, this protocol avoids the need to configure and maintain independent caching frameworks for service discovery purposes.

6.1.2 Network Support for Multimedia Applications using Netlets

Solutions to the Problem of Reservation Gaps using Netlets: The unpredictable behaviour of traffic within the non-QoS path segments present along a QoS-flow's path and the inability to support reservations across them can cause problems in providing end-to-end service guarantees in the Internet. I have described a

Netlets based approach to build a robust end-to-end QoS support model. I also proposed routing enhancements (*MR-S* and *S-MR*)[1] that when employed at QoS-nodes select a path for the QoS-flow with the minimum number of reservation gaps. Our technique features excellent dynamics and scales for large networks and user populations.

The good dynamics make support of short lived QoS-flows feasible. The control traffic generated (to monitor and manage the non-QoS path segments) is confined to the corresponding reservation gap, thus reducing congestion and packet loss. Overall, our solution provides a mechanism to support robust end-to-end QoS support in heterogeneous network environments such as the Internet. The technique described here makes it possible to deploy applications in the network which have quite hard QoS guarantee requirements, even when a significant number of network nodes support only best-effort service. Such techniques will be of critical importance in ensuring the graceful transition of the Internet from a best-effort service model to a service model featuring QoS guarantees.

Transparent Client-Server Selection using Netlets: I proposed a novel technique to support transparent and flexible server selection in the Internet. The Netlets based approach provides a client-side server selection solution which is server-customisable, scalable and fault transparent. Our approach combines the benefits of anycast addressing with a mechanism allowing the adoption of any server selection algorithm.

By using Netlets, service decision points can be deployed dynamically to the locations in the network where they can most efficiently serve a large number of clients. This approach makes our solution inherently scalable, since it minimises the amount of overhead generated by measurement probes. Overall, this approach demonstrates the versatility of implementing server selection algorithms that can work on the client-side of the network.

[1] jointly with my colleague Karol Kowalik [8]

Multicast Emulation using Netlets and Unicast (MENU): I proposed a new multicast protocol referred to as MENU. MENU builds a scalable multicast protocol model by pushing the tree building complexity to the edge network, thereby eliminating processing and state storage in the core of the network. MENU also provides reliable multicast communication services by supporting data caching within the network. Another architectural feature of MENU is that it automatically supports heterogeneous receivers; the Netlet nodes can perform media thinning within the network to suit end user terminals. It was shown through simulations that the resulting system provides an efficient means to incrementally build a source customisable secured multicast protocol which is both scalable and reliable. Furthermore, results showed that MENU employs minimal processing and reduced state information in networks when compared to existing IP multicast protocols. Results from a MENU prototype built using Java demonstrate that it is feasible to deploy such an architecture in today's IP networks.

Transparent QoS support of Network Applications using Netlets: I proposed a novel approach based on Netlets to transparently retrofit QoS support to legacy network applications. This approach is not restricted to a single QoS model or signalling protocol. Thus it may continue to be used even if the QoS support provided by the underlying network changes. This approach can be used as a permanent long-term solution to interface network applications with emerging QoS models on demand. Adapting this approach insulates future network applications from the specifics of the network support for QoS.

6.2 Future Work

This work demonstrated that network programmability can be introduced into existing networks using Netlets in an incremental and a cost-effective manner. Furthermore, a wide range of multimedia applications that benefits from Netlet

support was presented. To deploy Netlets on a large scale, some key practical issues need to be addressed. They are:

Resource Control: In traditional networks, bandwidth is usually the primary network resource that is shared among traffic flows at a network node. However, in active networking both node and link level resources will have to be managed. These resources typically include memory for incoming Netlets, CPU cycles, bandwidth etc. In order to ensure fair access and to avoid abuse of networking resources, Netlet nodes will have to impose strict limits on resource usage by third-party services.

I believe that RSVP like resource reservation mechanism could be used to build a flexible resource control framework for active networks. In this, I expect that a service setup phase using a resource-broker Netlet[2](similar to the flow setup phase in RSVP), can be initiated by end users who wish to launch services at distributed points within the network. This resource-broker Netlet, embedded with the required set of credentials (this should include the specification of the service to be installed and the permissions required for proper operation of the service) can then migrate to those distributed points and perform negotiations for service deployment.

Safety and Security: A major impediment to the wide spread deployment of any mobile code based systems is concern over safety and security. For example, in the above case of the resource broker Netlet, I assume that the network nodes and Netlets mutually trust each other. However, such a condition is not possible in a distributed shared infrastructure such as the Internet. Hence, robust safety and security mechanisms will have to be present to protect network nodes from malicious Netlets and vice-versa.

[2]I believe the resource broker Netlet would function in a way similar to mobile agents that are employed for buying/selling goods in electronic markets.

The current implementation of Netlets relies completely on the Java language to ensure safety and security. Future work should consider integrating robust mechanisms to support such features into the Netlets architecture. I have started investigating mechanisms to prevent impersonation attacks by malicious Netlet services. This model employs a single symmetry key, based on keyed hashes [64] for authentication purposes. However, this model addresses only an element of the larger problem. Considerable extra work will be required to make the Netlets architecture secure, although many aspects of the problem are being tackled in related projects on active networks and mobile agents.

Wider Applicability of Netlets: This thesis only considered the applicability of Netlets to support multimedia applications in the Internet. However, there are other emerging fields such as grid computing and peer-to-peer computing that are expected to benefit from Netlet support in the network. Below I consider grid computing as a candidate area for future Netlet applications.

A grid is a collection of geographically dispersed computing resources, providing a large virtual computing system to users [173]. Grid environments span wide area networks of heterogeneous computing resources, characterised by distinct administrative domains, and diverse operating systems and architectures.

Some of the fundamental requirements for applications that operate on grid environments are (i) to select the optimal locations for processing of a given data set; (ii) to select paths with desired QoS levels; and (iii) to scalably and reliably disseminate the given data set to those selected distributed resources. The protocols proposed in this thesis to support multimedia applications can easily be extended to work in grid environments. In our future work, we expect to adapt these protocols to suit such environments.

6.3 Concluding Remarks

Network programmability provides a way of introducing new or enhanced protocols in a network without requiring low-level programming access to network hardware. Active networking allows such programmability to be in response to user demand and thus offers a powerful paradigm for the development of new network services. This thesis has argued the case for using a mobile agent based architecture to deliver on the promise of active networks. A new mobile agent architecture called Netlets has been developed and its value has been demonstrated by implementing novel network services using Netlets. Further development of the Netlets architecture will result in a powerful tool for the development of network services for tomorrow's Internet.

Appendix A

An Analytical Model to Study the Reactive Service Deployment

A.1 Analytical Model

In this chapter, I present an analytical model to study the dynamics of a reactive service deployment model as observed by end user applications.

A.1.1 Reactive Deployment

Recall that in the reactive model, data packets carry the name of the Netlet service, which should process them at intermediate network nodes. Ideally it should be possible to discover and integrate the required service within a predefined time bound such that the overall delay experienced by packets is not excessive. Hence, it is crucial to study: (i) the delay experienced by packets requesting new services; and (ii) the deployment time involved to dynamically integrate a service into a Netlet node.

A.1.2 State Transitions at a Netlet Node

A packet arriving at a Netlet node will find the node either in:

(a) processing state: the active service is available and is processing packets; or

(b) dormant state: the active service is available but not processing packets; or

(c) idle state: the service is not available at the node; or

(d) discovery state: the node is in the process of discovering the required service.

The relevant state transitions are shown in Fig. A.1.

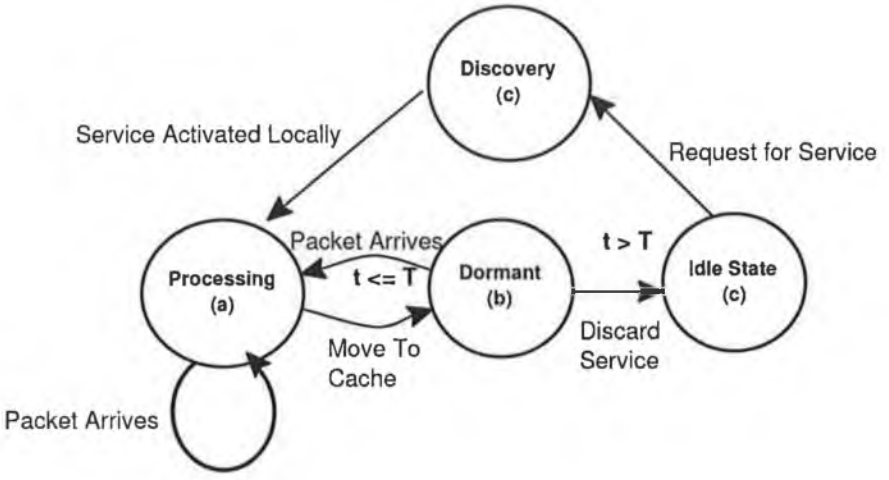


Figure A.1: State Transitions at a Netlet Node

A.1.3 M/G/1 Model with Setup Time

The Netlet node can be considered as a single input single output queuing system. The M/G/1 queuing model can be employed to study the delay involved in dynamically deploying a service at a Netlet node. In the M/G/1 system, arriving requests are modelled as a Poisson process with rate λ, while the processing times for jobs in the system are modelled as a general distribution. An M/G/1 model with vacation and setup times is used [174] (see Table A.1). The vacation and setup times correspond to discarding Netlet services and discovery of services in a Netlet network environment. Hence, results from such a model can be extended to study the pull based system.

Active Node	M/G/1 Queueing Model
Service is Active and Processing	System is serving requests
Service Unavailable	Vacation
Service Discovery	Setup

Table A.1: Netlet Node states and the M/G/1 queuing model with vacation and setup times

Netlet Node Model

The mean arrival rate of packets is assumed to be λ. The first and second moment of the processing time for packets at Netlet node are denoted by $E(B)$ and $E(B^2)$ respectively. Furthermore, the service processing time for a packet is modelled by a general distribution. The first and second moment of the service discovery time are represented by $E(T)$ and $E(T^2)$.

Its also assumed that a dynamically added Netlet service resides at a node for a period of T time units. A service which is inactive for more than T time units is removed from the Netlet node to ensure resource availability. For the system to be stable, we require that

$$\rho = \lambda E(B) \leq 1 \tag{A.1}$$

where, ρ is the utilisation factor of the service. Note, for the purpose of simplicity we only assume a Netlet node which, supports a single Netlet service. Furthermore it is assumed that the system serves the incoming packets on a First In First Out (FIFO) basis.

Mean Service Discovery and Deployment Time

Let the discovery and deployment time be denoted by t_{dd}. Note, when the requested service resides at the node, $t_{dd} = 0$. Hence, expected discovery and deployment time E(T) = 0. The condition to be satisfied for $E(T) = 0$,

$$t_{dd} = 0, \quad \forall \quad t \leq T \tag{A.2}$$

where t is the interarrival time between requests for the Netlet service and T is the cache invalidation time.

When packet arrival is exponentially distributed, probability that an incoming packets refers to a service that is available at the node is,

$$P_{request} = 1 - e^{-\lambda T}, \quad \forall \quad t \leq T \tag{A.3}$$

Thus, a packet not requesting a node resident service is,

$$P_{not-request} = e^{-\lambda T}, \quad \forall \quad t > T \tag{A.4}$$

From this, the expected mean discovery and deployment time,

$$E(T) = t_{dd} e^{-\lambda T} \tag{A.5}$$

Mean Waiting Time

The expected mean waiting for a packet at a Netlet node has the following elements:

- the residual service time, R_B, of the packet that is being currently processed; and
- the service time of all packets that are already in the queue; and
- the discovery and deployment delay, R_T, when in the case of the service was not available locally.

Using the mean value approach, the mean waiting time, $E(W)$, for a packet in such a system may be written as:

$$E(W) = E(N^q)E(B) + \rho E(R_B) + p_{idle}E(T) + p_{setup}E(R_T) \tag{A.6}$$

where, N^q = the number of packets in the queue;

$E(R_B)$ = the mean residual processing time;

$E(R_T)$ = the mean residual service discovery and deployment time;

p_{idle} = probability of the service not being present at the node; and

p_{setup} = probability of the node being in a service discovery phase;

where,

$$E(R_B) = \frac{E(B^2)}{2E(B)} \tag{A.7}$$

$$E(R_T) = \frac{E(T^2)}{2E(T)} \tag{A.8}$$

Based on the PASTA property [1], the period during the node is not processing packets involves an interarrival time followed by a service discovery time. Hence,

$$p_{idle} = (1-\rho)\frac{1/\lambda}{1/\lambda + E(T)} \tag{A.9}$$

Similarly it follows that:

$$p_{setup} = (1-\rho)\frac{E(T)}{1/\lambda + E(T)} \tag{A.10}$$

By substituting (A.9) and (A.10) into (A.6), we get,

$$E(W) = E(N^q)E(B) + \rho E(R_B) + \frac{1/\lambda}{1/\lambda + E(T)}E(T) + \frac{E(T)}{1/\lambda + E(T)}E(R_T) \tag{A.11}$$

Little's law state that

$$E(N^q) = \lambda E(W) \tag{A.12}$$

By using (A.12) and substituting into (A.11), we get

[1] It states that the fraction of customers who find, upon arrival, that the system is in some state A is exactly the same as the fraction of time the system is in state A.

$$E(W) = \frac{\rho E(R_B)}{1-\rho} + \frac{E(T)}{1+\lambda E(T)} + \frac{\lambda E(T) E(R_T)}{1+\lambda E(T)}) \tag{A.13}$$

Substituting Equation (A.5) into (A.13), we get

$$E(W) = \frac{\rho E(R_B)}{1-\rho} + \frac{t_{dd} e^{-\lambda T}}{1+\lambda t_{dd} e^{-\lambda T}} + \frac{\lambda t_{dd} e^{-\lambda T} E(R_T)}{1+\lambda t_{dd} e^{-\lambda T}} \tag{A.14}$$

Equation (A.14) represents the upper bound on the mean waiting time for a packet at a Netlet node can be calculated. Note, the first component on the right hand side represents the mean waiting for packet in a M/G/1 system without setup times. Note this component is commonly referred to as the Pollaczek-Khinchin (P-K) Formula [175]. The other two terms illustrates the impact of cache validation time and service deployment delays on the overall performance of a Netlet node.

A.2 Summary

In this chapter, I presented an analytical model which demonstrates the dynamics of reactive service deployment as observed by end user applications.

Index

A

Active Engine, 35
Active IP, 28
Active Network Encapsulation Protocol (ANEP), 27
Active Network Node, 25
Active Networks approach, 12
Active Node Model, 28
Active Packet Classifiers, 159
Active Packet Model, 28
Active Services, 34
Agent, 38
ALAN, 34
AMNet, 32
ANTS, 30
Anycast, 92
Apache Web Servers, 134
ASP EE, 31
Autonomous Tree Building, 110

B

Berkeley Packet Filter (BPF), 57

C

CANES, 32
Code-Server Model, 53
Communities of Interest(CoI), 96, 106

D

DAN, 32
DARWIN, 34
Data Quality Analyser (DQA), 153
Differentiated services, 81
Director Netlet, 133
Director Netlet Services, 92
DNS-based Node Discovery, 70, 170

E

Execution Environment, 26

F

FilterSpec, 118

G

Gap management Netlet, 124
Gateway Netlet Nodes, 152
Global IP Anycast(GIA), 100, 106
GT-ITM, 141

H

Home node, 53
Hot Spot Delegates(HSD), 104
Hot Spot Nodes, 98, 143
Httperf, 135

I

IEEE P1520, 13
Integrated services, 81
IPv6, 1
ISP topology, 126

J
JAIN, 13
Java Native Interface, 157

L
Link Stress, 144
Load from source, 53

M
M-Netlet, 153
M0, 29
MENU, 103
Mobile Agent, 39
Mobile Agent Systems, 39
Mobile Code, 25
MR-S, 88, 128
Multicast Error Recovery Delay, 146
Multicast Forwarding State, 145
Multicast gain metric, 142
Multicast Netlet, 153
Multicast Session Table(MST), 108
MultiProtocol Label Switching, 81

N
Negative Acknowledgements (NACK), 112
Netlet Label, 54
Netlet Managed Gap, 123
Netlet Management Engine (NME), 56
Netlet Runtime Environment (NRE), 52
Netlets, 3, 46, 50
NetScript, 33
Network Services, 27
Network Simulator (ns), 165
Node Operating System (NodeOS), 26
non-QoS aware applications, 114
non-QoS island, 85

O
OSPF, 88

P
Packet Communication Engine (PCE), 57
PAN, 31
PANDA, 31
PARLAY, 13
PATH message, 85
Phop, 86
Proactive deployment, 55
Proactive Packet Processing, 52, 67
Proactive Service Deployment, 169
Programmable Node approach, 12

Q
Q-flows, 81
Q-nodes, 81, 84
Q-segments, 81
QoS-Support Information Table, 118

R
Reactive deployment, 55
Reactive Packet Processing, 52, 68
Reactive Service Deployment, 72, 165
Receiver Table, 108
Replication Netlet(RepN), 108
reservation gaps, 81
Resource Control, 180
Resource Reservation Protocol (RSVP), 1
RESV message, 87
RIP, 88
RSVP Signalling Netlet, 150

S
S-MR, 89, 128
Self-organising Caches, 74
Self-organising Services, 44
Single source multicast, 103
Smart Packets, 28
SNAP, 29

SNMP, 84
SoftSwitch Consortium, 13
Source Route Option, 88, 109
Stigmergy, 73, 167
Strong Mobility, 40
Switchware, 33

T
TAMANOIR, 31
TCP SYN, 95
Time and Location Specific Network Services, 44

U
U-Netlet, 153
Unicast Netlet, 153

V
Virtual Host, 94
Virtual Primary Server, 92
virtualisation, 13

W
Weak Mobility, 40

Z
Zero Cache Cooperation, 78

Bibliography

[1] S. Deering and R. Hinden. Internet Protocol Version 6 IPv6. *RFC 2460*, December 1988.

[2] Lixia Zhang, Stephen Deering, and Deborah Estrin. RSVP: A New Resource Reservation Protocol. *IEEE Network*, 7(5):8–18, September 1993.

[3] Deering S and et al. Multicast Routing in Datagram Internetworks and Extended LANS. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[4] OPENSIG Working Group, http://comet.columbia.edu/opensig/.

[5] Active Networks. http://www.darpa.mil/ato/programs/activenetworks.

[6] Nwana H.S. Software agents: An overview. *Knowlegde Engineering Review*, 11(3):205–244, Nov 1996.

[7] Martin Collier. Netlets: The future of networking? In *In Proceedings of IEEE International Conference on Open Architectures and Network Programming (OPENARCH)*, San Francisco, USA, April 1998.

[8] Kalaiarul Dharmalingam, Karol Kowalik, and Martin Collier. RSVP Reservation Gaps: Problems and Solutions. In *In Proceedings of IEEE International Conference on Communications (ICC)*, pages 1590–1595, Anchorage, Alaska, USA, May 2003.

[9] Thomas P. Brisco. DNS support for load balancing. *RFC 1794:*, http://info.internet.isi.edu/innotes /rfc/files/rfc1794.txt, April 1995.

[10] D.Andresen et al. SWEB: Towards a Scalable World Wide Web Server on Multicomputers. In *In Proceedings of 1Oth IEEE International Syrup. on Parallel Processing (IPPS)*, Honolulu, Hawaii, USA, April 1996.

[11] C. Partrige, T. Mendez, and M. Miliken. Host anycasting service. *RFC 1546*, November 1993.

[12] R. Engel et al. Using IP Anycast for Load Distribution and Server Location. Sydney, Australia, November.

[13] H. Miura and M. Yamamoto et.al. Server Load Balancing with Network Support: Active Anycast. *In Proceedings of the Second International Working Conference on Active Networks (IWAN)*, pages 371–384, October 2000.

[14] Ellen W. Zegura, Mostafa H. Ammar, Zongming Fei, and Samrat Bhattacharjee. Application-layer anycasting: a server selection architecture and use in a replicated Web service. *IEEE/ACM Transactions on Networking*, 8(4):455–466, 2000.

[15] S. Deering, C. Partridge, and D. Waitzman. Distance vector multicast routing protocol. *RFC 1075*, November 1988.

[16] S. Deering et al. Protocol independent multicast spare-mode (PIM-SM): Motivation and architecture. *Internet draft, kdraft-ietf-idmr-pim-arch-04l*, October 1996.

[17] A. Ballardie. Core based trees (CBT) multicast routing architecture. *RFC 2201*, September 1997.

[18] S. Banerjee and B. Bhattacharjee. Scalable Application Layer Multicast. In *In Proceedings of ACM SIGCOMM*, Pittsburgh, PA, USA, August 2002.

[19] David D. Clark. The Design Philosophy of the DARPA internet protocols. In *In Proceedings of ACM SIGCOMM*, pages 106–114, August 1988.

[20] G Koprowski. Emerging Uncertainty Over IPv6. *IEEE Computer*, pages 106–114, November 1998.

[21] D. A. Patterson et al. Recovery-Oreinted Computing (ROC): Motivations, Defintion, Techniques, and Case Studies. Computer Science Technical Report UCB//CSD-02-1175, University of California, Berkely, 2002.

[22] Nils Bjrkman, Yong Jiang, and Torbjrn Lundberg. The Movement from Monoliths to Component-Based Network Elements. *IEEE Communications Magazine*, 2001.

[23] Tennenhouse D et al. Virtual Infrastructure: Putting Information Infrastructure on the Technology Curve. *Computer Networks and ISDN Systems*, 26(13), October 1996.

[24] M. C. Chan, J.-F. Huard, A.A. Lazar, and K.-S. Lim. On realizing a broadband kernel for multimedia networks. In *In Proceedings of 3rd Workshop on Multimedia Telecommunication and Applications*, pages 25–27, Barcelona, Spain, November 1996.

[25] Jit Biswas et al. The IEEE 1520 Standards Initiative for Programmable Network Interfaces. *IEEE Communications Magazine*, 36:64–70, October 1998.

[26] D. G. Andersen, H. Balakrishnan, and M. F. K. R. Morris. Resilient Overlay Networks. In *In Proceedings of Operating Systems Review*, page 131145, Alberta, Canada, December 2001.

[27] G. Goldszmidt and Y. Yemini. Delegated Agents for Network Management. *IEEE Communications Magazine*, 36(3):96–70, March 1997.

[28] Vijak Sethaput et al. Regatta: A Framework for Automated Supervision of Network Clouds. In *In Proceedings of IEEE International Conference on Open Architectures and Network Programming (OPENARCH)*, Anchorage, Alaska, USA, April 2001.

[29] Franco Travostino. Towards an Active IP Accounting Infrastructure. In *In Proceedings of IEEE International Conference on Open Architectures and Network Programming OPENARCH*, Tel-Aviv, Israel, March 2000.

[30] VPNC Virtual Private Network Consortium, http://www.vpnc.org/.

[31] Rebecca Isaacs. Lightweight, Dynamic and Programmable Virtual Private Networks. Tel-Aviv, Israel, March 2000.

[32] William La Cholter et al. IBAN: Intrusion Blocker Based on Active Networks. In *In Proceedings of DARPA Active Networks Conference and Exposition (DANCE)*, pages 182–188, San Francisco, CA, USA, May 2002.

[33] A. Hess, M. Jung, and G. Schaefer. FIDRAN: A Flexible Intrusion Detection and Response Framework for Active Networks. In *In Proceedings of IEEE International Symposium on Computers and Communications (ISCC)*, Antalya, Turkey, July 2003.

[34] V. C. Van. A Defence Against Address Spoofing Using Active Networks. *M.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T*, May 1997.

[35] Samrat Bhattacharjee, Kenneth L. Calvert, and Ellen W. Zegura. Self-Organizing Wide-Area Network Caches. In *In Proceedings of IEEE INFOCOM*, pages 600–608, San Francisco, CA, USA, April 1998.

[36] U. Legedza and J. Guttag. Using Network-Level support to Improve Cache Routing. *Computer Networks and ISDN Systems*, 30(22):2193–2201, 1998.

[37] Pradeep Sudame and B.R. Badrinath. Transformer Tunnels: A Framework for Providing Route-Specific Adaptations. In *In Proceedings of USENIX Annual Technical Conference*, New Orleans, Louisiana, USA, June 1998.

[38] Elan Amir, Steven McCanne, and Randy H. Katz. An Active Service Framework and Its Application to Real-Time Multimedia Transcoding. In *In Proceedings of ACM SIGCOMM*, pages 178–189, Vancouver, B.C., Canada, September 1998.

[39] Michael Fry and Atanu Ghosh. Application Level Active Networking. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(7):655–667, June 1999.

[40] Jiani Guo, Fang Chen, Laxmi Bhuyan, and Raj Kumar. A Cluster-Based Active Router Architecture Supporting Video/Audio Stream Transcoding Service. In *In Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, New Mexico, April 2003.

[41] Theodore Faber. ACC: Active Congestion Control. *IEEE Network*, 12(3):61–65, August 1998.

[42] S. Bhattacherjee, K. L. Calvert, and E. W. Zegura. Congestion control and caching in CANES. In *In Proceedings of IEEE International Conference on Communications (ICC)*, pages 25–30, Atlanta, GA, USA, 1998.

[43] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[44] D. Lin and R. Morris. Dynamics of Early Detection. *In Proceedings of ACM SIGCOMM*, pages 127–137, September 1997.

[45] Niraj Prabhavalkar, Manish Parashar, and Prathima Agrawal. LGC: An Active Congestion Control Mechanism. In *In Proceedings of Active Middleware Services (AMS)*, New York, USA, August 2000.

[46] Michael Welzl, Alfred Cihal, and Max Mhlhuser. An Approach to Flexible QoS Routing with Active Networks. In *In Proceedings of Active Middleware Services (AMS)*, pages 75–83, July 2002.

[47] Li-Wei and Lehman H. Active Reliable Multicast. In *In Proceedings of IEEE INFOCOM*, pages 581–589, San Francisco, USA, 1998.

[48] M. Sanders et al. Active Reliable Multicast on CANEs: A Case Study. In *In Proceedings of IEEE International Conference on Open Architectures and Network Programming (OPENARCH)*, Anchorage, Alaska, USA, 2001.

[49] Lidia Yamamoto and Guy Leduc. Building Bidirectional Multicast Trees Using Autonomous Relectors. In *In Proceedings of Internation Working Conference on Active Networks (IWAN)*, Tokyo, Japan, September 2000.

[50] Steven McCanne, Van Jacobson, , and Martin Vetterli. Receiver-driven layered multicast. In *In Proceedings of ACM SIGCOMM*, pages 117–130, August 1996.

[51] M Handley. An examination of MBONE performance. Technical Report Reserach Report-97-450, USC/ISI, 1997.

[52] Bernard Metzler, Till Harbaum, Ralph Wittmann, and Martina Zitterbart. AMnet: Heterogeneous Multicast Services based on Active Networking. In *In Proceedings of IEEE International Conference on Open Architectures and Network Programming (OPENARCH)*, New York, USA, March 1999.

[53] Ralph Keller et al. An Active Router Architecture for Multicast Video Distribution. In *In Proceedings of IEEE INFOCOM*, pages 1137–1146, Tel-Aviv, Israel, 2000.

[54] A Adl-Tabatabai, G Langdale G, Lucco S, and Wahbe R. Efficient and Language-independent Mobile Programs. In *In Proceedings of Conference on Programming Language Design and Implementation*, pages 127–136, Philadelphia, Pennsylvania, USA, May 1996.

[55] Active Networking Working Group. Architectural framework for active networks, Version 0.9. *http://www.cc.gatech.edu/projects/canes/arch/arch-0-9.ps*, July 1998.

[56] C. Gunter A. Jackson A. Keromytis D. Alexander, B. Braden and G. Minden. Active Network Encapsulation Protocol. *http://www.cis.upenn.edu/switchware/ANEP/docs/ANEP.txt*, 1997.

[57] David J. Wetherall and David L. Tennenhouse. The ACTIVE IP OPTION. In *In Proceedings of ACM SIGOPS European Workshop*, Conemera, Ireland, September 1996.

[58] Beverly Schwartz, Alden W. Jackson, W. Timothy Strayer, Wenyi Zhou, R. Dennis Rockwell, and Craig Partbridge. Smart packets: applying active networks to network management. *ACM Transactions on Computer Systems*, 18(1):67–88, 2000.

[59] Jonathan Moore, Michael Hicks, and Scott Nettles. Practical Programmable Packets. In *In Proceedings of IEEE INFOCOM*, pages 41–50, Anchorage, Alaska, USA, April 2001.

[60] Michael Hicks and Angelos D. Keromytis. A Secure PLAN. In *In Proceedings of Internation Working Conference on Active Networks (IWAN)*, Berlin, Germany, July 1999.

[61] Albert Banchs et al. Multicasting Multimedia Streams with ActiveNetworks. *ICSI technical report*, 97-050.

[62] D. Wetherall John Guttag and David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *In Proceedings of IEEE International Conference on Open Architectures and Network Programming (OPENARCH)*, 1998.

[63] D. Wetherall John Guttag and David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. *IEEE Network Magazine*, July 1998.

[64] R. Rivest. The MD5 Message-Digest Algorithm. *RFC 1321*, April 1992.

[65] Stephen J. Garland, Erik L. Nygren, and M. Frans Kaashoek. PAN: A High-Performance Active Network Node Supporting Multiple Mobile Code Systems. In *In Proceedings of IEEE International Conference on Open Architectures and Network Programming (OPENARCH)*, New York, USA, March 1999.

[66] Steven Berson Robert Braden, Bob Lindell and Ted Faber. The ASP EE: An Active Network Execution Environment. In *In Proceedings of DARPA Active Networks Conference and Exposition (DANCE)*, pages 238–245, San Francisco, USA, May 2002.

[67] Jean-Patrick Gelas, Saad El Hadri, and Laurent Lefevre. Towards the Design of an High Performance Active Node. *Issue of the Parallel Processing Letters (PPL)*, 13(2):149–167, June 2003.

[68] Vincent Ferreria, Alexey Rudenko, Kevin Eustice, V. Ramakrishna Richard Guy, and Peter Reiher. PANDA: Middleware to Provide the Benefits of Active Networks to Legacy Applications. In *In Proceedings of DARPA Active Networks Conference and Exposition (DANCE)*, pages 319–326, 2002.

[69] D. Decasper and B. Plattner. DAN - Distributed Code Caching for Active Networks. In *In Proceedings of IEEE INFOCOM*, pages 609–616, San Francisco, CA, USA, April 1998.

[70] Thomas Fuhrmann, Till Harbaum, Marcus Schller, and Martina Zitterbart. AMnet 2.0: An Improved Architecture for Programmable Networks. In *In Proceedings of International Working Conference on Active Networks (IWAN)*, Zurich, Switzerland, December 2002.

[71] K. Calvert, E. Zegura, and J. Sterbenz. CANEs: A Modest Approach to Active Networking. In *IEEE Computer Communications Workshop*, Phoenix, Arizona, USA, September 1997.

[72] Advanced Intelligent Networks. Bell communication research, inc. *AIN Release 1: Service Logic Framework Generic Requirements, FA-NWT-001132.*

[73] D. Scott Alexander et al. The SwitchWare Active Network Architecture. *IEEE Network Special Issue on Active and Controllable Networks*, 12(3):29–36, June 2000.

[74] S. de Silva, Y. Yemini, and D. Florissi. The netscript active network system. *IEEE Journal on Selected Areas in Communications (JSAC)*, 19(3):538–551, March 2001.

[75] Prashant Chandra et al. Darwin: Customizable Resource Management for Value-Added Network Services. In *IEEE International Conference on Network Protocols ICNP*, Austin, Texas, USA, October 1998.

[76] McCanne et al. Towards a common infrastructure for multimedia-networking middleware. In *In Proceedings of the Fifth International Workshop on Network and OS Support for Digital Audio and VideoNOSSDAV*, St. Louis, MO, USA, May 1997.

[77] Dickon Reed et al. Xenoservers: Accountable Execution of Untrusted Porgrams. In *In Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HOTOS-VII)*, Arizona, USA, March 1999.

[78] Danny Raz and Yuval Shavitt. Active Networks for Efficient Distributed Network Management. *IEEE Communications Magazine)*, 38(3):138–143, March 2000.

[79] W. Stallings. SNMP, SNMPv2 and CMIP. The Practical Guide toNetwork Management Standards. *Addison Wesley*, 1993.

[80] D. Larrabeiti, M. Caldern, A. Azcorra, and M. Uruea. A practical approach to Network-based processing. In *IEEE International Workshop on Active Middleware Services (AMS)*, Ediburgh, Scotcland, UK, July 2002.

[81] G. Hjalmtysson. The Pronto Platform: A Flexible Toolkit for Programming Networks using a Commodity Operating System. In *IEEE International Conference on Open Architectures and Network Programming (OPENARCH)*, Tel-Aviv, Israel, March 2000.

[82] T. Harbaum, A. Speer, R. Wittmann, and M. Zitterbart. Amnet: Efficient heterogeneous group communication through rapid service creation. In *In Proceedings of the Active Middleware Workshop (AMS)*, Pittsburgh, Pennsylvania, USA, August 2000.

[83] B. Krupczak, M. H. Ammar, and K. L. Calvert. Implementing protocols in Java: The price of portability. In *In Proceedings of IEEE INFOCOM*, pages 765–773, San Francisco, CA, USA, March 1998.

[84] Parulkar G, Plattner, and Decasper D. A Scalable, High Performance Active Network Node. *IEEE Network*, 31(1):8–19, January 1999.

[85] Werner Bux, Wolfgang Denzel, Ton Engbersen, and Ronald Luijten. Technologies and building blocks for fast packet forwarding. *IEEE Communications Magazine*, 39(1):70–77, January.

[86] A Kay. *Computer Software*, volume 251. Scientific American, 1984.

[87] T Selker. Coach: A teaching agent that learns. *Communications of the ACM*, 37(7):92–99, 1994.

[88] D. Lange and M. Oshima. Programming and deploying java mobile agents with aglets. *Addison Wesley, http://www.trl.ibm.com/aglets*, 1998.

[89] http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html.

[90] http://www.generalmagic.com/technology/odyssey.html.

[91] M. Breugst, L. Hagen S. Choy, M. Hoft, and T. Magedanz. Mobile Agent-Based Services in Fixed and Mobile Telecommunications Environments. *Software Agents for Future Communication Systems: Springer-Verlag*, pages 326–357, 1998.

[92] D. Wong et al. Concordia: An Infrastructure for Collaborating Mobile Agents. In *First International Workshop on Mobile Agents, Lecture Notes in Computer Science*, volume 1219, pages 86–97, Berlin, Germany, 1997.

[93] J.E. White. Telescript technology: The foundation for the electronic marketplace. *Technical Report, General Magic, Inc.*, 1994.

[94] L. Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, 1995.

[95] Neeran Karnik and Anand Tripathi. Agent Server Architecture for the Ajanta Mobile-Agent System. In *In Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA*, Las Vegas, NV, USA, 1998.

[96] http://www.agentbuilder.com.

[97] Holger Peine and Torsten Stolpmann. The Architecture of the ARA Platform for Mobile Agents. In *First International Workshop on Mobile Agents (MA)*, volume 1219, pages 50–61, Berlin, Germany, 1997.

[98] R. Gray, G. Cybenko, D. Kotz, R. Peterson, and D. Rus. D'Agents: Applications and performance of a mobile-agent system. *Applications and performance of a mobile-agent system. Software: Practice and Experience*, 2001.

[99] http://www.cs.uit.no/dos/tacoma/index.html.

[100] Vu Anh Pham and Ahmed Karmouch. Mobile software agents: An overview. *IEEE Communications Magazine*, 36:26–37, July 1998.

[101] G. Cugola, C. Ghezzi, G. Picco, and G. Vigna. Analyzing mobile code languages,mobile object systems. *Lecture Notes in Computer Science, Springer-Verlag (D)*, pages 94–109, Feb 1997.

[102] W. Die, P. C. Van-Oorschot, and M. J. Weiner. Authentication and Authenticated Key Exchanges. *In Proceedings of In Designs, Codes, And Cryptography*, 2(2):107–125, 1992.

[103] Digital signature standard. *National Institute of Standards and Technology. Digital Signature Standard. Technical Report FIPS-186, U.S. Department of Commerce*, May 1996.

[104] R Wahbe, S Lucco, and T Anderson. Efficient Software-Based Fault Isolation. In *In Proceedings of the 14th ACM Symposium on Operating Systems PrinciplesACM SIGOPS*, pages 203–216, North Carolina, USA, December.

[105] Necula and P Lee. Safe Kernel Extensions without Run-Time Checking. In *In Proceedings of the 2nd Symposium on Operating System Design and ImplementationOSDI*, pages 229–243, Seattle, USA, October 1996.

[106] Bennet S. Yee. A Sanctuary for Mobile Agents. In *Secure Internet Programming*, pages 261–273, New York, USA, 1999.

[107] T. Sander and C.F. Tschudin. Protecting Mobile Agents against Malicious Hosts. In *In Proceedings of Mobile Agents and Security*, volume 1419.

[108] Andrzej Bieszczad, Bernard Pagurek, and Tony White. Mobile Agents for Network Management. *IEEE Communications Survey*, 1998.

[109] A Chavez, A Moukas, and P Maes. Challenger: A Multi-Agent System for Distributed Resource Allocation. In *In Proceedings of International Conference on Autonomous Agents*, pages 323–331, New York, USA, 1997.

[110] Di Caro G and M Dorigo. AntNet: A Mobile Agents Approach to Adaptive Routing. Technical Report Technical Report IRIDIA/97-12, Universit Libre de Bruxelles, Belgium, 1997.

[111] T. White, A. Bieszczad, and B. Pagurek. Distributed Fault Location in Networks Using Mobile Agents. In *In Proceedings of the Workshop on Intelligent Agents for Telecommunications Applications (IATA)*, Paris, France, July 1998.

[112] Cameron Ross Dunne. Using Mobile Agents for Network Resource Discovery in Peer-to-Peer Networks. *SIGecom Exchanges, Newsletter of the ACM Special Interest Group on E-Commerce*, 2:1–9, 2001.

[113] A Chavez, A Moukas, and P Maes. Mobile Agents for Routing, Topology Discovery, and Automatic Network Reconfiguration in Ad-Hoc Networks. In *In Proceedings of IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, pages 200–206, Alabama, USA, April 2003.

[114] Ingo Busse, Stefan Covaci, and Andr Leichsenring. Autonomy and Decentralization in Active Networks: A Case Study for Mobile Agents. In *In Proceedings of Internation Working Conference on Active Networks (IWAN)*.

[115] Icrio Satoh. A Mobile Agent-Based Framework for Active Networks. In *In Proceedings of IEEE Systems, Man and Cybernetics ConferenceSMC*, October 1999.

[116] Stamatis K. Agent Populated Active Networks. 2000.

[117] D. Tennenhouse. Proactive computing. *Communications of the ACM*, 43(5):43–50, May 2000.

[118] PicoJava-II, Micro Architecture, http://www.sun.com/microelectronics.

[119] Sollins K and Masinter L. Functional requirements for uniform resource names. *RFC 1737, http://www.cis.ohiostate.edu/htbin/rfc/rfc1737.html*, December 1994.

[120] S. McCnne and V. Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *In Proceedings of the USENIX Coneference*, pages 259–269, Santa Fe, New Mexico, USA, 1993.

[121] Katz D. Ip router alert option. *Request for comments, RFC 2113*, February 1997.

[122] Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.

[123] Erik Guttman. Service location protocol: Automatic discovery of IP network services. *IEEE Internet Computing*, 3(4):71–80, 1999.

[124] S.V.Fuller, T.Li et al. Classless Inter-Domain Routng (CIDR): An Address Assignment and Aggregation. *RFC 1519*, 1993.

[125] D. Clark R. Braden and S. Shenker. Integrated services in the internet architecture: An overview. *RFC 1633*, June 1994.

[126] D. Clark R. Braden and S. Shenker. Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification. *RFC 2205*, September 1997.

[127] S. Blake et. al. An Architecture for Differentiated Services. *RFC 2475*, Dec.1998.

[128] A. Viswanathan E. Rosen and R. Callon. Multiprotocol Label Switching Architecture. *IETF RFC 3031*, January 2001.

[129] E. Rosen, A. Viswanathan, and R. Callon. Requirements for traffic engineering over mpls. *Internet draft, draft-ietf-mpls-traffic-eng.00.txt*, October 1998.

[130] E. Crawley. A framework for QoS-based routing in the internet. *RFC 2386*, August 1998.

[131] Y. Bernet et. al. A Framework for Integrated Services Operation over Diffserv Networks. *RFC 2998*, Nov.2000.

[132] Hermann de Meer, Jan-Peter Richter, Antonio Puliafito, and Orazio Tomarchio. Tunnel agents for enhanced Internet QoS. *IEEE Concurrency*, 6(2):30–39, 1998.

[133] G. Malkin. RIP Version 2. *RFC 2453*, November 1998.

[134] J. Moy. OSPF Version 2. *RFC 2178*, April 1998.

[135] M. Sayal and Y. Breitbart. Selection algorithms for replicated web servers. In *In Proceedings of Workshop on Internet Server Performance*, Madison, Wisconsin, USA, June 1998.

[136] Sandra G. Dykes, Kay A. Robbins, and Clinton L. Jeffery. An Empirical Evaluation of Client-Side Server Selection Algorithms. In *In Proceedings of INFOCOM*, pages 1361–1370, Tel-Aviv, Israel, 2000.

[137] R.L. Carter and M.E. Crovella. Server selection using dynamic path characterization in wide-area networks. In *In Proceedings of the IEEE INFOCOM*, Kobe, Japan, 97.

[138] Chad Yoshikawa and Brent Chun. Using smart clients to build scalable services. In *In Proceedings of USENIX 1997 Annual Technical Conference*, Anaheim, California, USA, 1997.

[139] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *In Proceedings of ACM SIGCOMM*, pages 97–110, Stockholm, Sweden, 2000.

[140] Paul Barford, Jin-Yi Cai, and Jim Gast. Cache placement methods based on client demand clustering. In *In Proceedings of IEEE INFOCOM*, pages 41–50, New York, USA, June 2002.

[141] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *In Proceedings of IEEE INFOCOM*, pages 126–134, New York, USA, 1999.

[142] Jose Santos et al. Understanding Service Demand for Adaptive Allocation of Distributed Resources. In *In Proceedings of IEEE Globecom*, Taipei, Taiwan, November 2002.

[143] Dina Katabi and John Wroclawski. A framework for scalable global IP-anycast (GIA). In *In Proceedings of SIGCOMM*, pages 3–15, Stockholm, Sweden, 2000.

[144] Christophe Diot and et al. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, 2000.

[145] Cheriton D.R Holbrook H.W. IP Multicast channels: EXPRESS support for large-scale single source applications. In *In Proceedings of ACM SIGCOMM*, pages 65–78, Cambridge, Massachusetts, USA, 1999.

[146] K. Yano and S. McCanne. The Breadcrumb Forwarding Service. *ACM SIGCOMM Computer Communication Review*, 30(2):41–49, April 2000.

[147] Bhaskar N. Source-specific protocol independent multicast. *Internet-draft*, pages draft–bhaskar–pim–ss–00.txt, 2000.

[148] Ion Stoica and T. S. Eugene Ng and Hui Zhang. REUNITE: A recursive unicast approach to multicast. In *In Proceedings of IEEE INFOCOM*, Tel-Aviv, Israel, 2000.

[149] Su Wen and et al. Building multicast services from unicast forwarding and ephemeral state. *Elsevier Journal of Computer Networks*, 38(3):327–345, 2002.

[150] J. Pansiot and D. Grad. On routes and multicast trees in the Internet. *ACM Computer Communication Review*, 28(1):41–50, June 1998.

[151] B.Riddle and A.Adamson. A quality of service api proposal. *http://apps.internet2.edu/qosapi.htm*.

[152] The WinSock Group. Windows sockets 2 protocol specific-annex. *http://apps.internet2.edu/winsock2.htm*.

[153] D. Florissi J. Zinky P. Wang, Y. Yemini. A Distributed Resource Controller for QoS Applications. In *In Proceedings of the IEEE/IFIP Network Operations and Management Symposium NOMS*, 2000.

[154] K. Nahrstedt and J. Smith. Design, Implementation and Experiences with the OMEGA End-point Architecture. *IEEE Journal on Selected Areas in CommunicationJSAC*, 17(7):1263–1279, September 1996.

[155] Jose Brustolonis. Quality of service support for legacy applications. In *International Workshop on Network and Operating System Support for Digital Audio and Video(NOSSDAV)*, New Jersey, USA, 1999.

[156] Nicola Ciulli et al. Qos provision to qos-unaware applications on intserv networks. In *INET*, 2000.

[157] G. Apostopoulos, R. Guerin, S. Kamat, A. Orda, and S. K. Tripathi. Intradomain QoS Routing in IP Networks: A Feasibility and Cost/Benefit Analysis. *IEEE Network*, 13(5):42–54, October 1999.

[158] Anees Shaikh, Jennifer Rexford, and Kang S. Shin. Evaluating the impact of stale link state on quality-of-service routing. *IEEE/ACM Transactions on Networking*, 9(2):162–176, April 2001.

[159] Apache Group. http://www.apache.org.

[160] David Mosberger and Tai Jin. Httperf: A Tool for Measuring Web Server Performance. In *First Workshop on Internet Server Performance*, pages 59–67, Madison, Wisconsin, USA, June 1998. ACM.

[161] Linux Kernel. http://www.kernel.org.

[162] Y.-H. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *In Proceedings of ACM SIGMETRICS*, pages 1–12, Santa Clara, Californa, USA, June 2000.

[163] Ellen W. Zegura et. al. How to model an internetwork. In *In Proceedings of IEEE INFOCOM*, San Francisco, USA, March 1996.

[164] Chalmers, R. and Almeroth, K. Developing a Multicast Metric. In *In Proceedings of IEEE Global Internet Conference, (GLOBECOM)*, pages 382–386, San Francisco, USA, November 2000.

[165] Data Quality Analyser, Anritsu Co-orp., http://www.anritsu.com.

[166] ARM Jazelle, http://www.arm.com/armtech/jazelle.

[167] Nazomi Jstar, http://www.nazomi.com.

[168] Kenneth L. Calvert, Jim Griffioen, Amit Sehgal, and Su Wen. Concast: Design and Implementation of a New Network Service. In *In Proceedings of Internation Conference on Networks and Protocols (ICNP)*, pages 335–342, Toronto, Canada, November 1999.

[169] E. W. Zegura Y. Chae and H. Delalic. PAMCAST: Programmable anymulticast for scalable message delivery. In *IEEE International Conference on Open Architectures and Network Programming (OPENARCH)*, New York, USA, June 2002.

[170] C. Cook, K. Pawlikowski, and H. Sirisena. ComAN: A Multiple-Language Active Network Architecture Enabled via Middleware. In *In Proceedings of IEEE International Conference on Open Architectures and Network Programming (OPENARCH)*, New York, USA, 2002.

[171] Network Simulator, http://www.isi.edu/nsnam/ns/.

[172] V. Jacbson and M. Karels. Congestion avoidance and control. In *In Proceedings of ACM SIGCOMM*, Vancouver, B.C., Canada, August 1988.

[173] Ian Foster and C Kesselman. *The Grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers, San Francisco, CA., 1998.

[174] Miller L.W. Alternating Priorities in Multi-class Queue. *Ph.D. Thesis,Cornell University, Ithaca, N.Y., USA*, 1964.

[175] Dimitri Bertsekas and Robert Gallager. Data Networks. *Prentice Hall Inc*, pages 186–206, 1992.