Event Correlations in Active Networks By Using Active Filters

Yihong Shangguan

School of Computer Science
McGill University, Montreal
July, 1999

A thesis submitted to the
Faculty of Graduate Studies and Research
In partial fulfillment of the requirements for the degree of
Master of Science

Résumé

Avec l'évolution de l'ordinateur moderne et les technologies resautiques, les systèmes d'ordinateurs et les applications devient de plus en plus complexes et dynamiques. L'un des problèmes des systèmes complexes et dynamiques et la difficulté de gestion de changement. La gestion de réseaux comprend la surveillance et contrôle, son but et la détection et le traitement des erreurs, la chute de performance et les fraude. La gestion centralisée des réseaux limite sérieusement la classification de la gestion de réseaux. La faiblesse de la gestion centralisée apparaît durant les périodes de lourdes congestion quand les interventions de gestion sont particulièrement plus importantes. Par conséquent, pour résoudre les problèmes (tel que passif, statique, rigide, etc.) causés par la centralisation, la fonction de la gestion du réseau doit être décentralisée et doit devenir plus active et flexible. Dans cette thèse, nous utilisons les agents mobiles et les technologies des réseaux actives pour implanter un outil de la gestion du réseau. C'est un filtre actif qui pourrait être très utile dans les plusieurs activités de la gestion du réseau tel que la gestion de ressources en temps réel; reroutage du flux en fonction de la charge; filtrage dans un réseau avec ou sans fil, dependant de l'application; et multicast en temps réel.

Abstract

With the development of modern computer and network technologies, computer systems and applications become more and more complex and dynamic. One of the problems of complex and dynamic systems is the difficulty of management of changes. Network management comprises of network monitoring and control, its aims include the detection and handling of faults, performance inefficiencies and security compromises. Centralized network management seriously limits the scalability of network management. The shortcomings of the centralized approach show up during the periods of heavy congestion when management intervention is particularly important. Therefore, in order to cope with problems (such as passive, static, rigid, etc.) arising from centralization, the network management functionality must be decentralized and should become more active and flexible. In this thesis, we use mobile agent and active network technologies to implement a network management tool, active filter, which could be very useful in various network management activities, such as, real-time resource management; load-sensitive flow rerouting; application-specific filtering in wireline/wireless network; and real-time multicast.

Acknowledgements

I am most grateful to my thesis supervisors Professor Petre Dini and Professor Monty Newborn for being excellent advisers on both my thesis and career, I would like to thank for their guidance, advice, and encouragement throughout the research. This thesis benefited from their careful reading and constructive criticisms.

I truly thank CRIM (Centre de recherche informatique de Montreal). It provides not only an cozy research environment for me, but also a great financial support for this research. I can not achieve anything without its support.

Many, many people have helped me in the preparation of this thesis. I would like to thank all of the team for the collaboration in the design and implementation of application. I benefited greatly from the formal and informal discussions with them.

I wish to thank the School of Computer Science for the graduate courses and the research environment. Thanks to Franca Cianci, Vicki Keirl, Josie Vallelonga, Lise Minogue, and Lucy St-James, for easing the procedure of dealing with the school.

I would like to thank my dear parents whose continual love and support throughout my whole life have kept me going.

Finally, special thanks to Hui Zhang for his deep love, support and encouragement during my study.

Acronym

AFL	Application Level Framing
AN	Active Networking
ANEP	Active Network Encapsulation Protocol
ANTS	Active Node Transfer System
CDs	Compact Discs
CBR	Constant Bit-Rate
CS	Communication Service
DAE	Distributed Agent Environment
DATs	Digital Audio Tapes
FIPD	Frame-Induced Packet Discarding
GMIB	Group Management Information Base
GOP	Group of Pictures
HeiTS	Heidelburg Transport System
IIOP	Internet Inter-ORB Protocol
IP	Internet Protocol
MASIF	Mobile Agent System Interoperability Facility
MbD	Management by Delegation
MDU	Media Data Unit
MIB	Management Information Base
MMC	Multimedia Multicast Channel
MPEG	Moving Picture Expert Group
NMS	Network Management Station
NVN	Netscript Virtual Network
OMG	Object Management Group
PDU	Packet Data Unit
PLAN	Packet Language for Active Networks
RCC	Routing Control Center
RCI	Router Control Interface
RMI	Java's Remote Method Invocation
RSVP	Resource Reservation Protocol
RTCP	Real-Time Control Protocol
RTP	Real-time Transport Protocol
TLV	Type/Length/Value
VBR	Variable Bit-Rate
VL	Virtual Link
VNE	Virtual Network Engine

Table of Contents

Résumé

Abstract

Acknowledgments

Acronym

1 Introduction
1.1 Network Management and Its Architecture 1
1.2 Centralized Network Management and Its Drawbacks 2
1.3 Management by Delegation (Mobile Agent Approach) 3
1.4 Active Network Approach 4
1.5 Active Filters 5

2 Background Knowledge of Filters
2.1 Filter Definition and Classification 7
2.1.1 Static Filter 8
2.1.2 Nomadic Filter 8
2.1.3 Modifiable Filter 9
2.1.4 Rigid Filter 9
2.2 The Heterogeneity and Resource Utilization Problems 9
2.2.1 The Heterogeneity Problem 10
2.2.2 Resource Utilization 11
2.2.3 Dynamic Quality of Service 12
2.3 Related Research in Filter Areas 12
2.3.1 Media Scaling and the Heidelberg Transport System – IBM ENC 13
2.3.1.1 Transparent and Non-transparent Scaling 13
2.3.1.2 Continuous and Discrete Scaling 13
2.3.2 The Multimedia Multicast Channel – UC, San Diego 15
2.3.2.1 Filters 15
2.3.2.2 Filter Propagation 15
2.3.2.3 Network as a Processor 16
2.3.3 Frame-Induced Packet Discarding – UC, San Diego 16
2.3.4 Dynamically Scaled Multimedia Streams – Sun Microsystems 16
2.3.5 Resource Reservation Protocol (RSVP) – Internet Engineering Task Force 17
2.3.6 Real-time Transport Protocol (RTP) – Internet Engineering Task Force 17
2.3.7 QoS-Filtering Model in Distributed Multimedia Application – Lancaster University 18
2.3.7.1 The QoS-Filtering Model 18

2.3.7.2 Model Entities..... 19
2.3.7.3 Negotiation..... 21
2.3.7.4 Stream Establishment..... 22
2.3.7.5 Stream Management, Monitoring and Maintenance..... 23
2.4 Filter Services and Mechanisms..... 25
2.4.1 Filter Services..... 25
2.4.1.1 End-to-End Scaling..... 26
2.4.1.2 In-line Adaptation..... 26
2.4.1.3 In-line Translation..... 27
2.4.2 Filter Mechanisms..... 27
2.4.2.1 Frame Dropping..... 27
2.4.2.2 Codec Filters..... 27
2.4.2.3 Color Reduction Filters..... 28
2.4.2.4 DCT-Filters..... 28
2.4.2.5 Mixing and Splitting Filters..... 28
2.4.3 Filter Location..... 30
2.4.4 Special Notes..... 31
2.5 Media Compression Technologies 31
2.5.1 MPEG 1..... 31
2.5.2 MPEG 2..... 32
2.5.3 Video Stream Data Hierarchy..... 32

3 Active Adaptation by Mobile Agent
3.1 Active Adaptation by Mobile Agent..... 35
3.1.1 Mobile Agent and Its Advantages..... 36
3.1.2 Mobile Agent Technology in Distributed Multimedia Systems..... 39
3.1.2.1 Passive Approach..... 39
3.1.2.2 Mobile Agents to the Rescue..... 40
3.2 Towards Active Network..... 41
3.2.1 Programmable vs. Active Network..... 42
3.2.2 Introduction to Active Networks..... 43
3.2.3 Active Network Concepts..... 44
3.2.3.1 Smart Packets..... 44
3.2.3.2 Active Nodes..... 45
3.2.4 Active Networks and Programming Interfaces..... 46
3.2.5 Active Network Design Models..... 47
3.2.5.1 Design Space Axis..... 47
3.2.5.2 Towards a Common Programming Model..... 49
3.2.6 Brief Overview of Current Active Network Technologies..... 52
3.2.6.1 PLANet & SwitchWare (University of Pennsylvania)..... 52
3.2.6.2 ANTS (MIT)..... 53
3.2.6.3 Netscript (Columbia University)..... 54
3.2.6.4 Smart Packet (BBN Technologies)..... 55
3.2.7 ANEP (Active Network Encapsulation Protocol)..... 55

4 Active Filters
4.1 Protocol Classification in Active Networks..... 58
4.1.1 Filtering Protocol Class..... 59
4.1.2 Combining Class..... 60
4.1.3 Transcoding Class..... 61
4.1.4 Network Management Class..... 61
4.2 Existing Active Filter Research..... 62
4.2.1 Active Networking and Congestion Control..... 62
4.2.1.1 An Architecture for Active Networking..... 62
4.2.1.2 Programmable Congestion Control..... 64
4.2.1.3 Application and Mechanisms of Congestion Control to MPEG..... 65
4.2.1.4 Limitations..... 66
4.2.2 Intellegent Communication Filtering..... 67
4.2.3 Firewalls..... 69
4.2.4 On-line Auctions..... 69

5 Extended Use of Active Filters in Other Domains
5.1 Active Filters in Real-time Resource Management..... 72
5.1.1 Motivation..... 73
5.1.2 Active Filter Architecture..... 74
5.1.3 Active Filter Runtime Environment..... 77
5.1.4 Active Filter Set Up..... 78
5.1.5 Implementation..... 78
5.2 Load-sensitive flow rerouting via active filters..... 82
5.3 Active Filters in Wireline/Wireless Network..... 85
5.3.1 Active Filters..... 85
5.3.2 Active Node..... 86
5.3.3 Media Scaling..... 86
5.3.4 Resource Probing and Automatic Teardown..... 88
5.3.5 Media Scaling Operations..... 88
5.4 Active Filters in Real-time Multicast..... 90
5.4.1 Active Node and Multicast Strategic Point..... 91
5.4.2 Active Filter Approach..... 92
5.4.3 Mechanisms..... 94

6 Simulation by Using Grasshopper
6.1 Distributed Agent Environment.....
6.1.1 Agents..... 99
6.1.2 Agencies..... 99
6.1.3 Regions..... 100
6.2 Communication Concepts..... 102
6.2.1 Multi-protocol Support..... 102
6.2.2 Location Transparency..... 102
6.3 Simulation by Using Grasshopper..... 103

7 Evaluations and Conclusions
7.1 Evaluations of Active Fitlers in Real-time Resource Management and Load-Sensitive Flow Rerouting..... 105
7.2 Evaluation of Active Filters in Wireline/Wireless Network..... 106
7.3 Evaluation of Active Filters in Real-time Multicast..... 107

Bibliography..... 108

Appendix Source Code of Active Filter Application
1 Agent.java..... 112
2 FilterController.java..... 112
3 FilterDaemon.java..... 113
4 FilterServer.java 114
5 SelectivePacketDropper.java 116
6 BandwidthCheckAgent.java..... 120

List of Figures

2.1	Static Filter	8
2.2	Nomadic Filter	8
2.3	Window of Insufficient Resources	11
2.4	Localized Dynamic Control	24
2.5	MPEG Video Stream Data Hierarchy	33
2.6	Picture Sequences in Display and Video Stream Order	34
3.1	Not-So-Smart Packet	45
3.2	Smart Packet	46
3.3	ANEP Packet Format	57
5.1	Active Filter Network Model	75
5.2	Active Node Architecture	76
5.3	Testbed Topology	79
5.4	Resource Trees	84
5.5	Filter Control and Filter Agent Interactions	87
5.6	Media Scaling Process	88
5.7	Active Node in Multicast Tree	91
5.8	Multicast Tree with Different Bandwidth Properties	92
5.9	Filter Assumption (Si = So + Sdrop)	95
5.10	Internal Data Flow in an Active Node	96
5.11	ACKs' Aggregation	97
5.12	Single Input/Output Active Node	98
6.1	Hierarchical Component Structure	100
6.2	Multi-Protocol Support	102
6.3	Grasshopper Testbed	103
6.4	Agency 1, Agency 2 and Agency 3	104
6.5	GUI of BandwidthCheckAgent	104

List of Tables

2.1	Classification of Filters	9
2.2	Generalization of Filter Systems	24
2.3	Filter Mechanisms	29
3.1	Program Encoding Technologies	50
3.2	Operating System Technologies	51

Chapter 1
Introduction

With the development of modern computer and network technologies, computer systems and applications become more and more complex and dynamic. Internet is a typical example, it is a huge network, composed of millions of heterogeneous computers connected through a wide variety of network links, and numerous kinds of applications. Furthermore, hosts, network links and applications are added or removed constantly. Each application can be envisioned as a dynamic system living in an ever-changing environment. One of the problems of complex and dynamic systems is the difficulty of management of changes.

1.1 Network Management and Its Architecture

Network management comprises of network monitoring and control, its aims include the detection and handling of faults, performance inefficiencies (e.g., high latency delays), and security compromises (e.g., unauthorized access). To accomplish these goals, management application do the following:

- Collect real time data from network elements, such as routers, switches, and workstations. For example, they collect the number of packets handled by the given interface of a router.
- Interpret and analyze the data collected. For instance, they may recognize security events, such as repeated illegal attempts to login on a workstation.
- Present this information to authorized network operators, possibly by displaying a map of current traffic.
- Proactively react, in real time, to management problems, possibly by disabling a link that is experiencing faults.

The architectures of network management systems are categorized as follows:

- **Centralized network management**: A single centralized Network Management Station (NMS) overlooks the management. It queries the network components on a timely basis to determine the health of the network.
- **Hierarchical network management**: A single centralized NMS is aided by a set of subordinate NMSs. The subordinate NMSs take off some of the responsibilities of the central NMS.
- **Peer network management**: A set of NMSs manage the different domains of the network with timely interaction amongst them.
- **Fully distributed network management**: A totally distributed management architecture in which a large number of NMSs perform the management by using specialization, delegation, cooperation, etc.

Currently, the prevalent network management architecture is centralized network management, which is achieved by having management stations routinely poll the managed devices for data, looking for anomalies. But this kind of centralized management architecture has many drawbacks as we will describe in the following section.

1.2 Centralized Network Management and Its Drawbacks

Centralization seriously limits the scalability of network management. As the dimension of the network grows, the management station has to communicate with a larger number of devices, and to store and process an ever increasing amount of data. This leads to the need for high cost hardware dedicated to the management station, to poor performance, or even to the impossibility to cope with the dimension of the network. The area of the network around of the management station experiences heavy traffic due to the combination of messages sent around by the management station and those containing data from the devices. The shortcomings of the centralized approach show up during the periods of heavy congestion, when management intervention is particularly important. In fact, during these periods:

- The management station increases its interactions with the devices and possibly downloads configuration changes, thus increasing congestion.

- Access to devices in the congested area becomes difficult and slow (sometimes even impossible), and
- Congestion, as an abnormal status, is likely to trigger notifications to the management which worsen congestion.

A further problem with polling is that a component can suffer multiple state changes in less than one round-trip time. Therefore, it is essential that network management employs techniques with more immediate access and more ability to scale.

Similar problems also affected routing table computation when it was centralized. A Routing Control Center (RCC) gathered information on network topology, calculated the routing table for each router in the network, and downloaded it into proper device. The heavy traffic load in the area around the RCC and the difficulty of management areas far from RCC led to the development of distributed routing.

Therefore, in order to cope with problems (such as passive, static, rigid, etc.) arise from centralization, the network management functionality must be decentralized and become more active and flexible, for example, the complex diagnosing and information gathering activities can be moved from the management station into the network. Many researches have been done in this area, what we interest in this thesis are two technologies: *mobile agent* [CHK97] and *active networks* [TW96].

1.3 Mobile Agent Approach

Mobile agent represents a clear effort towards decentralization and increased flexibility of network management functionality. Mobile agents can be used for a variety of purposes in network management, they could provide the following advantages [SC]:

- **Distribution of management code**. Mobile agents are used to distribute the code to the managed network elements when necessary, instead of moving large amount of data to the manager over the network, this reduces substantially bandwidth usage and reduces the network bottleneck as well as makes the architecture more scalable.
- **Decentralization**. They are effectively used to decentralize network management activities.

- **Dynamic changing of network management policies**. As the network environment grows and changes the policies need to be changed over time and also in order to tackle temporally changing problems the management policies need to be altered. Instead of going through rewrite, compile and run cycle, the management policies are dynamically changed by writing new agents easily without altering the provided infrastructure.
- **Monitoring and statistics**. Mobile agents are suitable for retrieving large number of samples of network management variables i.e. suitable for monitoring of these variables and also for studying the behavior of network components over long period of time. They could be used for network components monitoring and for gathering statistics.
- **Data collection**. They are suitable for data collection, searching and filtering.
- **High speed networks**. They are suitable for high speed network management in which case it is not practical to bring all the network data to the manager.

Generally speaking, mobile agent is a new technology which can overcome many limitations showed in centralized network management and the traditional client/server architecture. Detailed information about mobile agent will be given in Chapter 3.

1.4 Active Network Approach

Active network [TW96] has recently attracted a lot of attention. The idea is that instead of having packets be passive entities that are carried around, packets can be active and change the behavior of the network. Generally speaking, an active network is one where node functions can be openly and dynamically programmed. Software will be loaded and executed in intermediate nodes. This software could be developed and deployed by anyone. This would permit any vendor to introduce novel protocols that support innovative functions in network nodes, as much as they do for end nodes. This software, packaged as mobile agents, called *Smart Packet* [KMH+98], could be dynamically dispatched and activated by network providers or users. Hence, it enables the creation of self-configuring, self diagnosing and self-healing networks. This involves actions, such as

alarm and event reporting, accounting, configuration management, workload monitoring, etc.

Active network technology can improve network management. For example, management centers can send programs to the managed nodes, which can bring the following advantages as:

- the information content returned to the management center can be tailored (in real-time) to the current interests of the center, thus reducing the traffic as well as the amount of requiring examination.
- many of the management rules employed at the management center can now be embodied in programs which, when sent to managed nodes, automatically identify and correct problems without requiring further intervention from the management center.
- Smart Packets shortens the monitoring and control loop -- measurements and control operations are taken during a single packet's traversal of the network, rather than through a series of `set` and `get` operations from a management station.

1.5 Active Filters

As we mentioned above, mobile agent and active network do have many advantages which could be applied to network management, especially the management of changes. Therefore, in this thesis, we try to use mobile agent and active network technologies to implement a network management tool, *active filter*, which could be very useful in various network management activities, such as:

(1) Real-time resource management;
(2) Load-sensitive flow rerouting;
(3) Application-specific filtering in wireline/wireless network, and
(4) Real-time multicast.

The rest of the thesis is organized as follows: Chapter 2 describes the background knowledge of filters; Chapter 3 devotes to basic concepts of mobile agent and its applications in network management, followed by detailed descriptions on active network

technologies, the research groups and different approaches; Chapter 4 describes the existing research on the applications of active filters; Chapter 5 gives the explicit design and implementation of the extended use of active filters in other domains, which includes network resource management area, application-specific filtering in wired/wireless network area, and real-time multicast area. Chapter 6 describes the simulation part by using mobile agent platform: Grasshopper. The thesis closes with the evaluations and conclusions of our research.

Chapter 2
Background Knowledge of Filters

In this chapter, we discuss the background knowledge of filters that either influenced the research presented in this thesis or is crucial to the understanding of the filter mechanisms and important filter concepts. Section 2.1 gives the filter definition and classification. Section 2.2 describes the heterogeneity and resource utilization problems. Section 2.3 discusses the related research in filter areas, which includes: media scaling and the Heidelberg Transprot System; the multimedia multicast channel; frame-induced packet discarding; dynamically scaled multimedia streams; resource reservation protocol (RSVP); real-time transport protocol (RTP); and QoS filtering model in distributed multimedia applications, etc. Filter services and mechanisms are presented in Section 2.4. In Section 2.5, we introduce some media compression technologies.

2.1 Filter Definition and Classification

In our work, a *filter* is a software only or hardware supported, object that implements some actions on its inputs based on a set of rules. For example, a filter could operate within the network or at the network edge to process continuous media streams or packets to satisfy the requirements of the distinct receivers of that particular stream or packets, or to adapt quickly to the whole network changing conditions.

The rules on which a filter takes action could relate to:

- *QoS (quality of service) parameters*: bandwidth, jitter, throughput, delay, etc.
- *Alarms*;
- *Management commands*: set, get, etc.

A filter could be static or nomadic, the set of rules on which it bases could be rigid or modifiable. Therefore, any combination leads to a particular filter type.

2.1.1 Static Filter

A *static filter* is a filter that is built into a network element by the ISP (Internet Service Provider) to perform its functions based on one or more given rules, and it is not movable (which means that it cannot move from one network element to another). For example, a router is a kind of static filter, because it can route packets to their respective destinations based on the packet's header information (See Figure 2.1).

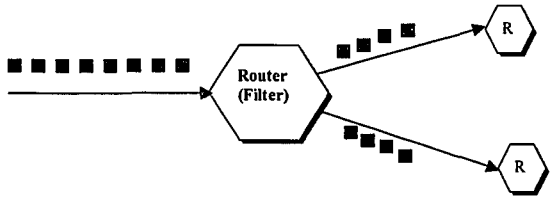


Figure 2.1 Static Filter

2.1.2 Nomadic Filter

A *nomadic filter* could move from one network element to another. For example, it could be a mobile agent (or Smart Packet) created by users, and be sent to one or more network

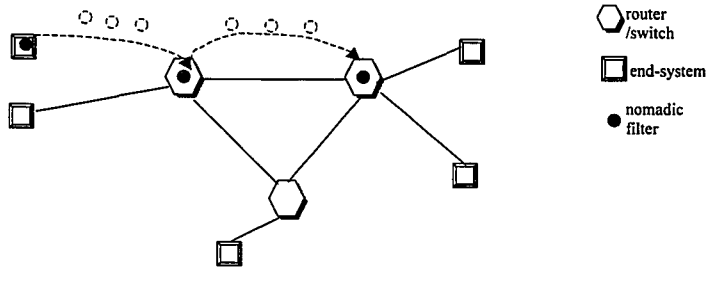


Figure 2.2 Nomadic Filter

elements (such as switch, router, or even server) to perform the various filtering functions. Nomadic filter could be downloaded into a network element, and execute its functions for one time then move to another network element (See Figure 2.2).

2.1.3 Modifiable Filter

A *modifiable filter* means the set of rules on which it takes actions are modifiable. For example, a filter based on QoS parameters encapsulates a QoS negotiation and renegotiation (which means the changing of QoS parameters according to the application's requirements and the network conditions) to achieve a receiver's QoS requirements and promote optimum system utilization. A modifiable filter can be static or nomadic.

2.1.4 Rigid Filter

A *rigid filter* means the set of rules on which it takes actions are rigid, not modifiable. A rigid filter can also be static or nomadic.

We summarize the classification of filters by Table 2.1:

Categories	*Movable*	*Modifiable*
Static rigid filter	no	no
Static modifiable filter	no	yes
Nomadic rigid filter	yes	no
Nomadic modifiable filter	yes	yes

Table 2.1 Classification of Filters

2.2 The Heterogeneity and Resource Utilization Problems

As we have already given the definitions and classifications of filters, the question comes after is that why we need filters. In this section, we demonstrate the present degree of heterogeneity present at every level of modern distributed systems; heterogeneity in applications, end-systems and networks. This section continues with other driving reasons such as resource utilisation and dynamic QoS management.

2.2.1 The Heterogeneity Problem

As a result of information technology expanding into new areas of society and geographic location, there now exists a vast assortment of end-system and communica-tion architectures, not to mention the multitude of application software. Heterogeneity is totally acceptable in isolated systems; Once these systems are connected together, the problems of establishing a true open environment emerge. While the proliferation of the Internet Protocol (IP) has gone some way to solve the interconnection problems for data transfer, the issues relating to the transfer of real-time continuous media are still unresolved. This problem is particularly acute in distributed group applications where many disparate receivers are wishing to exchange continuous media data with each other despite capability and architectural differences. Generally speaking, the network heterogeneity problem shows in the following aspects:

User and Application Requirements

The range of applications and user requirements is likely to be quite diverse. For example, in multimedia applications, the perception of video and audio quality is user-dependent and hence users may express different requirements in playout qualities. This will be encompassed in the specification of distinct QoS requirements by disparate users.

End-system Capabilities

Considering end-system hardware, heterogeneity is present in: CPUs, I/O devices, storage capabilities, compression support (dedicated boards/software), internal inter-connect architecture, communication protocol support, network interfaces, etc. These issues place limits on the end-system's capabilities to process, consume and generate multimedia data.

Networks

End-systems are likely to be connected to different networks which not only have different bandwidth capabilities but also varying access delay characteristics. For example: medium access control mechanism, maximum and minimum data unit size, service types, packet loss rates, propagation delays, congestion, etc.

2.2.2 Resource Utilization

As well as the heterogeneity problem there exists the ever-present problem of limited resources. Anderson [ATW+90] defined the window of insufficient resources (see Figure 2.3). Figure 2.3 shows the development of computer resources against the resource requirements of the various application domains. The implications of Figure 2.3 are that optimum resource utilization is an inherent requirement in distributed applications, as such services have a high demand for network bandwidth allocation, storage capacities and processor time. The rationale for using filters to reduce bandwidth required is that it is unnecessary to transmit data to receivers that either cannot use it or do not wish to use it. If fully and correctly implemented, filters can cut out the unwanted data at the earliest opportunity, or during network congestion, filters can drop less important data according to some set of rules, hence achieving optimum bandwidth utilization.

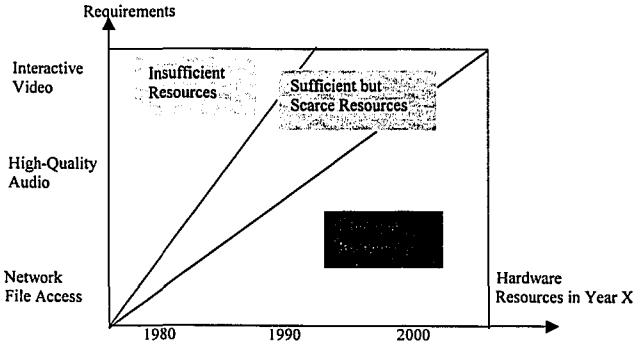


Figure 2.3 Window of Insufficient Resources [ATW+90]

Besides, limited resources on end-systems result in buffer overflows, lateness in processing data and inability to process data. If the processing of data can be distributed among a number of nodes, and some processing completed before it reaches an overloaded client, this reduces the demands on that client.

2.2.3 Dynamic Characteristics of Quality of Service

Solving the heterogeneity problem involves providing individual QoS, and achieving resource optimization implies accurate QoS levels; to realize both of these in a continually changing environment requires the ability to adapt and dynamically alter QoS levels.

The most important factor to consider when implementing a dynamic QoS mechanism is the **cost**, in terms of resources required and of changing the current QoS levels. To perform a complete end-to-end re-negotiation within a multiparty connection may take up to three signaling messages per receiver and may involve resynchronizing, i.e. initializing, and altering parameters of codec hardware. Therefore, the cost of end-to-end adaptation or scaling determines the frequency at which re-negotiation may take place and how dynamic may be the QoS monitoring and control system.

Filter operations could be one solution to make the localization of dynamic QoS control, which means fine adjustments to a client's received data rate can be made which require only interaction between a receiver and its closest filter agent. Localization of control offers a number advantages over end-to-end control: firstly, the propagation delay between client and flow control process may be much smaller thus allowing more accurate and reactive feedback control. The signaling messages between client and control nodes traverse fewer hops and are therefore have less impact on the total end-to-end bandwidth. Moreover, they have less chance of suffering the effects of a congested network (even if the signaling data is prioritized, it may still be delayed in a congested non-reservation based network).

2.3 Related Research in Filter Areas

The term filter has been used by many researchers each with their own interpretation of the word. Systems incorporating what can be viewed as filter operations generally fall into the categories concerning:

- end-to-end scaling
- media processing filters
- resource reservation

All approaches aim to support *adaptive and reactive distributed multimedia applications*; some goals are to solve heterogeneity in networks, or in end-systems; other aims are resource optimization and reservation.

2.3.1 Media Scaling and the Heidelberg Transport System - IBM ENC

The *Heidelberg Transport System (HeiTS)* [Hehmann91] is a communication system for real-time delivery of video and audio.

2.3.1.1 Transparent and Non-transparent Scaling

Transparent scaling is performed by the transport system independently of the upper protocol layers. It relies on the transport system being able to identify suitable stream segments to discard, such as individual frames. *Non-transparent scaling* involves an interaction between the transport system and upper layers. Generally, non-transparent scaling involves the altering of encoding parameters, on live streams, or the recoding of a stream before it is passed to the transport system. These application level modifications may be enacted as a result of congestion induced feedback from the transport system.

2.3.1.2 Continuous and Discrete Scaling

Scaling can be applied in one of two ways: either within a connection or sub-stream, or by adding and removing connections and sub-streams. This gives a choice between fine and coarse granularity hence these two approaches are called *continuous* and *discrete* scaling respectively.

Continuous scaling relies on two functions: scale-up and scale-down. These requests will either be performed by the transport entity or the sending application depending on whether scaling is being performed transparently or non-transparently, respectively. If a number of packets are late, or lost, the receiver assumes the received stream is suffering the effects of congestion and initiates the scaling procedure (this is also called receiver-initiated scaling). This procedure consists of three stages:

- The first reaction to congestion is 'local scaling', by discarding late packets. This does not affect the sender. It is only intended as a short term measure.

- If the number of lost or late packets exceeds a certain threshold a scale-down request is made to the sender to throttle back its traffic. The sender may be throttled back to zero transmission, but the connection is still maintained so scaling up can be quickly performed once the current congestion problem is relieved.
- If congestion continues and a number of scale-up attempts fail, a decision to terminate the connection and release resources is made.

There is no way for the transport entity to know when congestion has finished, so an attempt to scale-up is made after a certain amount of time. Care has to be taken to ensure that the sender does not scale-up too early or too late. Too early and the system may have to scale-down immediately, producing an oscillation; too late is not as critical but to make the best use of resources an optimum time must be established.

Discrete scaling is a connection oriented scaling method. That is, a stream is split into sub-streams and each of these is transmitted on a separate connection. Receivers then scale-up and scale-down by accepting or rejecting connections, and full quality is gained by receiving all sub-streams. Discrete scaling implies a minimum of feedback and hence each receiver has more control over its individual QoS received, as opposed to continuous scaling where, in a multicast session, each receiver may issue scale-down requests to the sender thus restricting the transmission to the ability of the least capable receiver.

The continuous and discrete scaling methods are in fact two very separate mechanisms. Continuous scaling does *not* work for multicast traffic because the least capable receiver will inhibit all other receivers in the same group. *Using intermediate nodes to down-scale and up-scale traffic could provide the solution to this.*

Discrete scaling is a way to solve the problems created by continuous scaling, i.e., to allow different receivers in the same multicast session to obtain different QoS levels. Complexities with discrete scale lie with resynchronising the various sub-streams. Discrete scaling gives strong support to hierarchical encoding schemes thus allowing heterogeneity in multipeer communications.

2.3.2 The Multimedia Multicast Channel - UC, San Diego

The *Multimedia Multicast Channel (MMC)* [PPA+92][PPA+93] is a communication abstraction akin to that of a cable TV channel. That is, receivers 'tune in' to a particular set of streams, or just one stream, to obtain a desired multimedia service. Filters may be instantiated to tailor a particular stream to a receiver's requirements if the transmitted stream is incompatible with the receiver. Filters thus allow a number of heterogeneous receivers to obtain different QoS levels from a common flow.

2.3.2.1 Filters

A filter is a transformer of one or more input streams of a multi-stream into an output stream, where the output stream replaces the input streams in the multi-stream. Filters are categorized into *selective*, *transforming* and *mixing* filters.

- The *selective filter* is the simplest form of filter. The filter only forwards certain segments depending on some criteria. Such a filter may perform frame rate reduction, by only forwarding segments from certain frames; or if the multi-stream contains a hierarchically coded stream the filter can choose only to pass the base layer, by discarding all segments from any enhancement layers.
- A *transforming filter* involves more processing. The filter performs some calculation or computation on the stream to produce a new stream. *A filter does not necessarily have to reduce bandwidth*: a decompression filter would be equally as valid.
- A *mixing filter* takes two or more streams and combines them into fewer streams than originally present.

2.3.2.2 Filter Propagation

Each filter is separately instantiated by individual receivers at the end of a complex dissemination tree. The power of filters on the MMC is their ability to *propagate* towards the source. If a filter produces a new stream that occupies less network bandwidth than its input stream, as many filters do, then relocating the filter closer to the source optimizes network usage.

Furthermore, by propagating a filter to a specialized network server it is possible to off-load some end-system processing. Filters also have the ability to combine together.

Combining filters reduces the processing incurred by intermediate network nodes. The mixed filter will then, if possible, continue to propagate towards the source.

2.3.2.3 Network as a Processor

By performing operations of different levels of complexity en route between source and receiver implies that the network may be treated as a processor. It is therefore possible to trade off bandwidth requirements with processing performed on network routers and gateways.

2.3.3 Frame-Induced Packet Discarding - UC, San Diego

Frame-Induced Packet Discarding (FIPD) [RRV93] is a method to improve network bandwidth utilization. The scheme involves an efficient frame dropping strategy based on discarding corrupted frames. A router implementing the FIPD scheme monitors loss on a stream. Once a certain percentage of the Packet Data Units (PDU) constituting a frame have been lost, the remaining PDUs belonging to that frame are then discarded. The *loss threshold*, termed the packet *resiliency*, is dependent upon the encoding scheme used. Some encoding schemes would not tolerate any PDU losses, implying that a single lost packet would require the whole frame to be discarded. In inter-coded compression schemes, such as MPEG, the effect of a corrupt frame may be propagated through to neighboring frames. For example, a corrupt I-frame may mean that the P- and B- frames dependent on it are also corrupted.

2.3.4 Dynamically Scaled Multimedia Streams - Sun Microsystems

The *Dynamically Scaled Multimedia Streams* [HSF93] concept is based on the use of hierarchically encoded media streams. These streams may be filtered at various network nodes, such as routers or transport relays, that are not explicitly aware of the stream semantics. Each media stream consists of a number of sub-flows. Filtering is achieved by discarding sub-flows at various points in the network. The Hoffman approach includes a syntax to describe scalable flows and extension of the media transport interface to understand this syntax. The objective of using scalable flows and filtering is to achieve congestion control, bandwidth and admission control, and receiver traffic selection. That is, the filter system can intelligently discard packets at the time of congestion; also, lower

capability links may only be able to receive a subset of the stream sub-flows, and an end-system may also select the quality it requires by only receiving some of the subflows.

The scalable stream syntax includes adding a time stamp, sync bit and packet sequence number to each Media Data Unit (MDU). Each MDU is also tagged to identify which sub-flow it belongs to. Based on this tag, two filter types are defined:

DiscardTagEqualTo(tag_val)
DiscardTagGreaterThan(tag_val)

Higher priority sub-flows are assigned lower tag values; control data and data that cannot be filtered are assigned zero. Where the data in a scalable flow consists of components which have an ordering, e.g. a base layer and further enhancement layers, the base layer will be assigned the lowest value. The DiscardTagEqualTo(tag_val) filter can be used to drop an individual sub-flow, whereas the DiscardTagGreaterThan(tag_val) filter will discard all sub-flows above the set value. For example, it could discard all enhancement layers above the base layer.

2.3.5 Resource Reservation Protocol (RSVP) - Internet Engineering Task Force

The *resource reservation protocol (RSVP)* [ZDE+93] has another perspective on the concept of filters. With the use of RSVP, clients may reserve resources (e.g. buffers) at switches and end-systems. There are a number of reservation styles, which are differentiated by the type of filter used. The filter mechanism determines which packets may use the reserved resources:

- *Fixed-Filter* This reservation is applicable to only one sender, i.e. only packets from the specified sender will be forwarded.
- *Shared-Explicit* This reservation allows multiple senders to use the same reserved resources, but only the senders that are explicitly specified by the receiver.
- *Wildcard-Filter* The filter allows all senders to use the reserved resources.

2.3.6 Real-time Transport Protocol (RTP) - Internet Engineering Task Force

The real-time transport protocol (RTP) [SCF+94] provides end-to-end transport functions suitable for distributed applications using continuous media. It is intended for unicast and

multicast network services. RTP is based on *application level framing* (AFL) and hence operates on top of existing transport protocols, primarily UDP. The real-time protocol specification consists of two parts: RTP for data transfer and the Real-Time Control Protocol (RTCP) for monitoring and distributing information on the current level of QoS transmitted and received on a session.

To support real-time data transfer the RTP protocol header has a number of important fields: *payload type*, *sequence number* and *time-stamp*. The payload specifies the media type encapsulated within the PDU, e.g. MPEG video, PCM audio, etc. The sequence number denotes the order in which the packets are transmitted from the source. The time stamp represents the time at which the data segment being transmitted was sampled. The timestamp can be used by a receiver to resynchronise data and to monitor packet arrival jitter. The sequence number can be used to monitor packet loss and recodering.

2.3.7 The QoS Filtering Model in Distributed Multimedia Applications – Lancaster University

Distributed multimedia applications include video and audio conferencing, dissemination and on-demand services. The requirements of these applications are very different but all have real-time considerations in terms of the transmission of multimedia. Multicast mechanisms allow a source to transmit data to a number of receivers simultaneously. A typical multipeer communication session may therefore consist of a number of one-to-one and one-to-many connections forming a many-to-many group communication.

2.3.7.1 The QoS-Filtering Model

The *QoS-filtering model* [YGH+96] involves placing filters at strategic points, such as network nodes, gateways, specialized servers, etc., around a multicast network tree. The designated source may then send at the quality required by the highest capability receiver while low capability receivers acquire a filtered down version of the media stream.

Filters in QoS-filtering model are objects which transform continuous media streams in some way. This may involve reduction of video frame rate, adjustments to presentation

quality or conversion to different compression formats. A filter may be a software only object or enjoy hardware support.

As the characteristics of the underlying network or the nature of the transmitted media change, filters may be added or removed from the multicast dissemination tree. Filter objects may also logically move around the current tree to achieve the optimum location of execution. This is known as *filter propagation* which we mentioned above. By implementing this approach all receiver's disparate quality requirements are satisfied. Filtering is one of the realistic solutions to heterogeneous QoS within multicast communications.

2.3.7.2 Model Entities

Within the QoS-Filtering Model there are a number of key objects that combine to build the overall QoS-Filtering Architecture. The *stream*, *source*, *filters* and *clients* are controlled and maintained by a *session manager*, although the clients do have some autonomy with respect to controlling local filter agents. The following gives the definitions of these related entities:

Client: The client object represents the communication data sink. Within a multipeer stream a number of clients will be associated with the end leaves of the dissemination tree. Clients are responsible for initiating a media service or joining an ongoing communication session. Service instantiation, or service joining, is achieved by clients issuing connect requests to a session manager.

Source Server: The source server centers around a daemon that waits for media requests from session manager. On receiving a connection request the daemon spawns a source agent. The source server is always in existence whether any source agents are transmitting media or not. The location and services available from the source server are well known to the session manager.

Source Agent: The source agent is the communication data source. The source agent resides at the root of a multipeer dissemination tree. The source agent responds to the request issued by a session manager by disseminating a message for set-up negotiation to filter server and clients.

Filter Server: The filter server is based around a daemon that waits for requests to instantiate filter mechanisms. The mechanisms may be instantiated at connection set-up time or a during an existing communication session. Like the source server, the filter server is ever-present and well known to the session manager. The filter server may reside on a switching node, an end system (source or sink), or on a specialized filtering node. A number of filter servers may exist along a communication path. If occupying a switch, the filter server would be integrated into the switching and forwarding mechanisms of the switch.

Filter Agent: The filter agent performs the filtering operations on a continuous media stream. Filter agents are spawned by the filter servers to perform some requested filtering operation. Depending on the filter operation complexity, a filter agent may be a software-only process or may be supported by specialized hardware.

Session Manager: The session manager is responsible for informing group members of services available, and of the QoS capabilities of those services. It is also required for dealing with client requests to initiate services, join existing service sessions and leave existing sessions. Clients may also request changes in the quality of their received service, which may involve the session manager interacting with the source agent and with a number of filter servers and filter agents to adapt the stream quality. Note that the clients can also renegotiate QoS levels with their closest filter agent. The session manager also maintains the current state of the group within communication session. There is one session manager per communication session.

The Stream: The stream is a data transmission between a source agent and multiple client objects. The flow of data through the stream follows the path of the established dissemination tree. The flow hides the identity of the clients from the source agent. The source agent is only aware of the session manager and the address to which it is sending, which may be a multicast group and/or a filter/switch object.

The Group: The group is a list of receivers and senders within a single multipeer communication session. The current quality that each client is receiving, and that the source is sending, are also held with the group membership information. The group information is held and maintained by the session manager, possibly by implementing a Group Management Information Base (GMIB).

2.3.7.3 Negotiation

This section describes the negotiation process in the QoS-Filtering model which is entailed before a continuous media data transmission is initialized. The negotiation includes the *reservation of resources* and *filter allocation policy*.

The initial negotiation consists of a three-way message-passing initiated by the client. Exchanges between clients, the session manager and the source are in the form of flow-specifications (FlowSpec). The FlowSpec characterizes quantitative QoS levels, actions on thresholds, reliability requirements, service commitment and appropriate filter operation identification. As the QoS of the flow may be changed, through filtering mechanisms, on its way to the client(s), each client supplies a client-FlowSpec describing its particular QoS requirements and characteristics.

Client Connect Request

The client issues a request to connect to a service. This FlowSpec request is issued to the session manager in the form of the service required, associated QoS specifications for throughput, delay, jitter and reliability and the service commitment for these QoS metrics. If valid, the request is processed and passed to the source server.

Resource Management: Admission Control and Filter Allocation

The source server, on receiving a FlowSpec connect request message, either accepts the QoS or changes it to a level it can supply. Instead of always lowering the proposed QoS level in the FlowSpec it is conceivable that the source server may actually increase the QoS levels it can supply. The FlowSpec is then sent to the client(s), containing the original or adjusted QoS parameters.

While this message is en route to the clients, each node in the intended path for the continuous media transmission performs an admission test based on available resources and the QoS parameters within the FlowSpec. If the QoS levels cannot be met, i.e. the required resources are not available, a filter is allocated to reduce the required QoS to match the level of resources available. If resource reservation is available, and required, the approximate amount of resource is reserved. The QoS levels within the FlowSpec are

then adjusted to reflect this change. The message is passed on to the next down-stream node until each node has allocated any required filters and reserved any resources necessary. Any node, including the source, can reject the connection request if insufficient resource is available. The connection may also fail if allocating a filter to reduce the required QoS levels breaches one or more of the other specified QoS levels.

Connection Agreement

Once the message has reached the clients, all required resources will have been reserved along the proposed paths and any filters necessary will have been provisionally allocated. The FlowSpec message encapsulates whether it was necessary to lower the proposed QoS characteristics. Each client has the option of whether to accept the final proposed QoS levels, lower the QoS requirement and allocate a filter, or reject the connection. If accepted the client confirms the connection an acknowledgment to the sender. This is relayed through each node, hence allocating the reserved resources and instantiating any filter operations needed. Any over-committed resources, such as if the client has lowered the QoS requirement, are relinquished. A reject results in the reserved resources being freed and filters deallocated.

Stream Characterization

Stream characterization is intended to give a client a reasonable approximation of the QoS to be expected from a given service. All information gathered is held by the session manager and relayed to the client objects as requested or at regular intervals.

2.3.7.4 Stream Establishment

Data transfer begins when the QoS negotiation, set-up stage is complete, i.e. as soon as the source receives the acknowledgment and desired source QoS output level from the clients. This QoS level requirement will be the FlowSpec from the highest capability receiver.

Client Application Initialization

Once the client has agreed to, and acknowledged, the proposed QoS levels it instantiates all receiving processes that are necessary. The receiving application then waits for the

data stream to arrive. If the connection setup fails the client applications may be either explicitly terminated by the session manager or may time-out.

Filter Server/Filter Agent Initiation

On the return acknowledgment path any filter agents that are required are spawned by the relevant filter server. The filter agents generally take a single stream in and generate the number of required filtered streams. The filter agents also retain any information essential to initialize a receiving process which is transmitted within the stream. This enables clients to join an ongoing stream at a later time by connection directly to a filter agent, i.e. without the need to involve the source agent. The filter agent informs the session manager of any changes it may experience.

Source Server / Source Agent Interaction

On receiving the client acknowledgment FlowSpec the source server spawns a source agent which is responsible for gathering, packetizing and sourcing the data stream. All control over the data stream is handed over to the source agent, the source server reverts to waiting for further service requests, as far as the current session is concerned the source server is no longer needed. The source agent informs the session manager about itself and the QoS levels it is transmitting; this information is intermittently updated.

2.3.7.5 Filter Propagation and Renegotiation

One fundamental aspect of the QoS filtering model is the ability for filters to propagate. That is changes in end-system loading and application/user requirements filters can be relocated within the dissemination tree. Filter propagation achieves optimum resource utilization. Filter propagation occurs when QoS levels of all outputs of a node are lower than the input QoS. Filters always propagate toward the source, thus reducing network loading, if the filter is a reduction filter. Filters may ultimately propagate to the source itself and hence reduce the output QoS of the source.

In practice, limits may be placed on the how far filters can propagate toward the source. For example, filters may only be allowed to propagate within the local LAN domain. A policy could also exist where any clients joining 'late' have to accept the quality available

locally. There exists a trade-off between flexibility and the overhead of providing such dynamic adaptability.

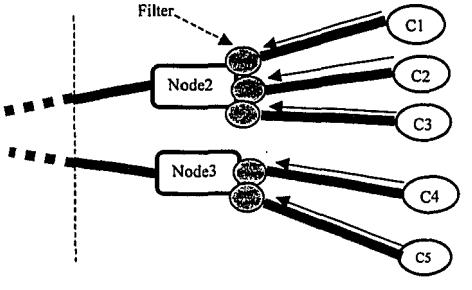


Figure 2.4 Localized Dynamic Control

Clients can, during the lifetime of a connection, issue requests to the session manager to renegotiate the whole stream. If the session manager receives enough renegotiate requests it may decide to initiate a complete stream renegotiation. It is just replaced by another connection with different QoS levels, resource reservations and filter allocations.

The characteristics of the seven systems introduced above can be summarized into the following Table 2.2:

Filter Systems	Developers	Characteristics
HeiTS	IBM ENC	• For real-time delivery of video and audio • Transparent and non-transparent scaling • Continuous and discrete scaling
MMC	UC, San Diego	• For delivery of a set of streams • Selective filter, transforming filter, mixing filter • Filter propagation
FIFD	UC, San Diego	• For delivery of frames • Be dependent upon the encoding scheme used (such as I-, B-, P- frames)

Dynamically Scaled Multimedia Streams	Sun Mcrosystems	• For delivery of multimedia streams • Based on the use of hierarchically encoded media streams • Filtering is achieved by discarding sub-flows.
RSVP	IETF	• Another perspective on the concept of filters : filter mechanism determines which packets may use the reserved resources • Fixed-filter, shared-explicit, wildcard-filter
RTP	IETF	• For delivery of continuous media for unicast and multicast
QoS-Filtering Model	Lancaster University	• For distributed multimedia applications • Place filters at network strategic points • QoS negotiation and renegotiation • Various filtering functions • Filter propagation

Table 2.2 Generalization of Filter Systems

From the Table 2.2 above, we can see all of the filters are static filters, which means they are not movable. The *filter propagation* which was proposed by UC (San Diego) and adopted by Lancaster University does not mean movable filters, it just means the instantiation of another filter which is nearer to the source and deactivate the filters on its downstream. Some of the filters are modifiable, such as QoS-Filtering Model (Lancaster University), FIFD (UC, San Diego) and Dynamically Scaled Multimedia Streams (Sun Microsystems).

2.4 Filter Services and Mechanisms

Based on the background knowledge of filter research, in this section we will outline filter services and filter mechanisms, filter location will also be discussed here. Filter services can be either explicitly requested by end-system applications or may be instantiated by underlying network entities. One or more filter mechanisms may be adopted in the process of delivering a filter service.

2.4.1 Filter Services

Filter services include: end-to-end scaling; in-line adaptation; in-line translation, etc.

2.4.1.1 End-to-End Scaling

There are several approaches to changing or adapting the traffic between the source and sink. The first method is end-to-end scaling, which can be considered as end-system or *source based filtering*. End-to-end scaling involves the source altering its own output to suit the requirements of the receiving parties.

Scaling may be performed to react to different conditions in the receiver end-system or because of congestion within the transmission medium.

The source system may be called upon to perform extra processing or some special adaptive function. Such operations include:

- changing sampling rates;
- quantization scales;
- adjusting luminance levels and other encoding parameters.

If the media to be distributed is already compressed when presented to the sending host, it is still possible for the sender to perform source based filtering. This will generally involve only transmitting a selection of a media stream. For example, if a client only requires mono audio then, by clever storage techniques, only one mono channel would be retrieved from the storage medium.

2.4.1.2 In-line Adaptation

In-line adaptation is the name given to simple filter operations that change a data stream's characteristics with a minimum of processing and involving no decompression. The stream or streams concerned are usually altered by discarding information. The mechanisms used to achieve this service can be relatively simple. Hence, it is possible for network nodes with limited processing power, e.g. switches, to perform such filtering.

Frame dropping filters and the *hierarchical splitting filter* are typical mechanisms that may be called upon to offer this service.

2.4.1.3 In-line Translation

In-line translation includes filter operations that require more processing complexity to convert media into a different form. These operations require an in depth knowledge of the encoding schemes used and how to interpret information contained within the compressed media. It is possible that these mechanisms may be performed on a separate network device in order to not to overload the switching nodes within the network. This type of service may be called upon if a receiver has very different requirements to other members of its own multicast group, or if the source and sender are quite simply incompatible. These operations can be very adaptive and maybe used as a reactive measure in the communication. Mechanisms included are: low-pass filtering, color reduction filtering, transcoding and mixing, etc.

2.4.2 Filter Mechanisms

One or more filter mechanisms may be adopted in the process of delivering a filter service which include: frame dropping filters; codec filters; color reduction filters; DCT-filters; mixing and splitting filters, etc.

2.4.2.1 Frame Dropping Filters

The *frame dropping class of filter* is a media discarding filter used to reduce frame rates. There are two types of frame dropper:

- *priority based frame dropper*: it has knowledge of the frame types and drops frames according to importance. It may be to used where interdependencies exist between frames (such as MPEG, it has I, B, P frames)
- *simple frame dropper*: it has less knowledge than the priority based frame dropper, and can be used where each frame in a stream is independently encoded (such as in Motion-JPEG or I-picture-only MPEG).

2.4.2.2 Codec Filters

They perform specific compression related operations.

- *Transcoding Filter*, converts data streams encoded in one particular compression standard into a different compression standard.

- *Compression/Decompression Filter*, where end-systems do not have the ability to deal with compressed media of any type, this kind of filter may be used. These filters perform compression or decompression on behalf of a source or client system.

2.4.2.3 Color Reduction Filters

They discard varying amounts of color information from a video stream, used where an end-system does not have a color display or has a limited display.

- *Color to monochrome filter*, has the effect of removing all color information from a video stream and leaving just luminance data.
- *DC-Color filter*, provides a means of reducing the bit-rate of a video stream by discarding only some of the color information.
- *Dithering Filter*, reduces the number of bits used to represent the color and luminance depth (i.e. bits-per-pixel) in the uncompressed image.

2.4.2.4 DCT-Filters

The DCT-based filters are specific to the sequential DCT compression schemes, such as JPEG, MPEG, and H.261.

- *Low-pass Filter*, removes the high frequency components from an image, thus reducing image quality but maintaining frame-rate.
- *Re-quantization Filter*, is a method of reducing bit-rate while maintaining the same frame rate.
- *Limiting Filter*, is based around a highly dynamic low-pass or re-quantization filter. The filter is designed to convert a variable bit-rate (VBR) data stream into a constant bit-rate (CBR) data stream.

2.4.2.5 Mixing and Splitting Filters

They provide the functions of combining many streams to one stream, and separating a single stream into a number of sub-streams, respectively.

- *Mixing Filter*, is used where the end-system only has the capability to decode one stream at a time or where some resource saving is possible by combining a number of streams. In certain cases the combined stream may have a lower bandwidth

requirement then the sum of the streams' separate bandwidth requirements. We have the following kinds of mixing filters:

- *Interleaving Frame Mixer*
- *Intra-Frame Mixer*
- *Video and Audio Multiplexer*
- *Audio Mixer*

- *Splitting Filter*, can perform reverse operations to the mixing filter and also two other appropriate uses:
  - *Individual QoS Splitter*: a splitting filter can be used to separate a mixed media stream in order to associate specific media dependent QoS parameters to the individual streams. For example, a MPEG System (video and audio) stream may be split into a single video stream and single audio stream.
  - *Hierarchical Splitter*: in situations where a source cannot provide a hierarchically encoded stream but such a stream would be advantageous, the splitting filter can take a single stream and split it into the required sub-stream structure. For example, an MPEG 1 video stream could be split so that the I, P, B pictures are assigned to separate sub-streams.

The various filter mechanisms can be generalized into Table 2.3.

Frame dropping filter	Priority based frame dropper Simple frame dropper	
Codec filters	Transcoding filter Compression/decompression filter	
Color reduction filter	Color to monochrome filter DC-color filter Dithering filter	
DCT-filter	Low-pass filter Re-quantization filter Limiting filter	
Mixing and splitting filter	Mixing filter	Interleaving frame mixer Intra-frame mixer Video and audio multiplexer Audio mixer
	Splitting filter	Individual QoS splitter Hierarchical splitter

Table 2.3 Filter Mechanisms

2.4.3. Filter Location

In the path between a source and a recipient, a filter operation can be performed either within the network (routers and switches) or at the network edge (i.e. end-systems and gateways). The criteria for determining this location depend on the following factors:

Data Unit Encapsulation

The amount of information a particular node knows about a stream, and hence its ability to execute a filtering function, relies on the way the data is encapsulated within the protocol data unit (PDU). For example, separate video frames can be encapsulated within separate PDUs, and so a gateway or router could performed a frame dropping function by dropping PDUs.

Switch/Router Capability

Implementing filtering within switches/routers will require reprogramming of the switch. It is plausible that switches/routers could be reconfigured and reprogrammed to incorporate filter operations.

End-system Capability

End-systems in certain cases are the optimum place for filters. The term 'end-systems' includes, sources and receivers, high level gateways (because these are also at the network edge). Certain sources or low end receivers may not have the necessary capabilities to execute a particular operation, that is, to execute it within the imposed time constraints. The operation may therefore be performed at a less optimal location in the network.

Available Bandwidth

If the manipulation of a stream involves a major change in the bit rate, such as compression or decompression, then in order to utilize the network resources to best effect, the operation must be executed where the stream will not cause adverse network loading. That is, an operation that produces a larger bit-stream should be executed as close to the receiver as possible and conversely and operation that reduces the size the bit-stream should be located as close to the source as possible.

Time Constraints

No matter how powerful a filtering engine may be, time will always be consumed. This affects both the transmission delay experienced, and the jitter. A trade-off has to be reached between the benefits of filtering, network performance and time constraints.

Filter Propagation

In a dynamic heterogeneous network the optimum location for a particular filter operation may change over time, hence a filter must have the ability to move or propagate to a more suitable node. For example, propagation may occur when a client joins or leaves a current session, or if a node or link becomes heavily loaded and some processing must be off-loaded to neighboring node.

2.4.4 Special Notes

Putting filters into switches/routers does have advantages, it is the most logical place, but this may be detrimental to the overall performance of the switch/router. End-system filtering is easier to implement and causes least disruption to existing services but does not realize the full potential of filtering operations.

2.5 Media Compression Technologies

Some of the filter algorithms described in later chapters operate on compressed media streams. These algorithms exploit certain characteristics of the stream syntax to simplify filtering. Hence, this section describes the common international standards on media compression.

Compression technology is primarily employed to reduce the amount of data required to represent text, graphics, audio and video. This leads to savings in storage space, improvements in access speeds and in distributed environments it leads to more efficient utilization of network bandwidth.

2.5.1 MPEG 1

Moving Picture Expert Group (MPEG) is responsible for the development of international standards for digitally coded motion video and its associated audio. The initial intention

was to provide a common representation and format for the encoded video and its associated audio on various digital storage media such as Compact Discs (CDs), Digital Audio Tapes (DATs), Winchester disk and optical drives. These devices are capable of providing a continuous transfer rate of about 1.5 Mbit/s. This standard is now typically referred to as MPEG 1. The features that MPEG 1 supports include forward playback, reverse playback, random access, fast search, error robustness, and some editing functionality.

2.5.2 MPEG 2

The MPEG group, in association with the Expert Group of ATM Video Coding of the ITU-T SG 15, has been developing MPEG 2 [MPEG 2, 94] which is targeted for very high quality coding of moving pictures and associated audio. The features supported by the MPEG 2 standard include: constant bit rate transmission, variable bit rate transmission, random access, channel hopping, scalable decoding, editing, as well as special functions such as fast forward playback, fast reverse playback, slow motion, pause and still pictures. MPEG 2 codecs are required to be backward compatible with MPEG 1 encoded bit stream.

Any MPEG stream conforms to a generic two layer structure:

- The *system layer* which contains timing and other information needed to multiplex, demultiplex and synchronize playout of the audio and video substreams.
- The *compression layer* which handles the compression and decompression of the audio and video streams.

2.5.3 Video Stream Data Hierarchy

MPEG encoder has a hierarchical structure. This hierarchy is illustrated in Figure 2.5 and described below.

The topmost level, the *sequence* layer, consists of a header, one or more groups of pictures and an end of sequence of marker. The information carried in the header is employed to initialize the state of the decoder. The *group of pictures* (GOP) layer contains a header with time and editing information plus a number of pictures. GOPs

represent the smallest coding unit that can be independently decoded within a sequence and that may form random access points.

The *Picture* structure represents a single frame of motion video. Three frame types are defined. I-pictures, or intra-coded pictures, are coded without reference to other pictures. They provide the access points and are moderately compressed. P-pictures, or forward

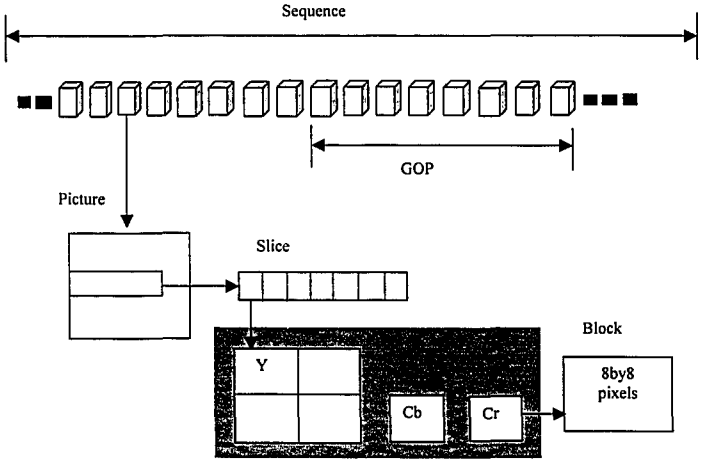


Figure 2.5 MPEG Video Stream Data Hierarchy
(Adapted from [Yeadon96])

predictive coded pictures, are more efficiently coded by employing motion compensation. These form reference points for further prediction. B-pictures, or bi-directionally predicted coded pictures, are coded by using motion compensated prediction from a past and a future I or P-picture. B-pictures demonstrate the highest degree of compression and are never used as reference points for further prediction.

Because of the inter-dependencies between the various picture types, the order in which pictures are transmitted, stored, or retrieved is not the same as the display order. Instead,

the pictures are arranged in the order in which they are required for decoding. This is illustrated in Figure 2.6. Each picture is composed of a header, containing ordering, picture type, and coding information, plus one or more *slices*. The *slice* structure is composed of a header and one or more macroblocks. The header contains position and quantizer scale information which may be used to recover from local corruption. If the bit stream becomes unreadable within a picture, the decoder may recover when the next slice arrives without having to drop an entire picture.

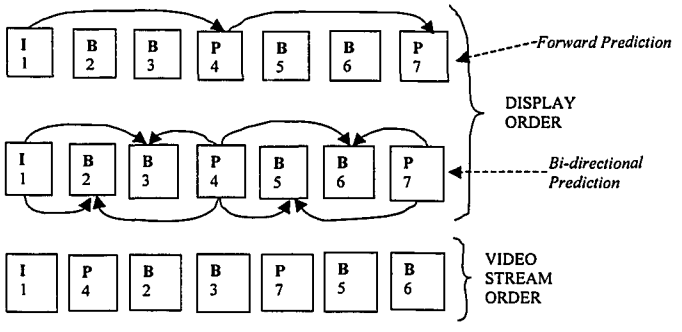


Figure 2.6 Picture sequences in display and video stream order

The *macroblocks* represent the basic unit for motion compensation and quantizer scale changes. Each macroblock structure contains a header a six 8 by 8 blocks: 4 blocks of luminance, 1 block of Cb chrominance and 1 block of Cr chrominance. The header contains quantizer scale and motion compensation information. The *blocks* represent the basic coding unit.

Chapter 3
Active Adaptation
by Mobile Agent and Active Network

In this chapter, we are going to introduce two active adaptation technologies: mobile agent and active network. *Mobile agents* are autonomous, intelligent programs that can migrate from one machine to another in a heterogeneous network. From the computation point of view, mobile agents co-locate data and computation by bring the computation to the data, rather than by bringing the data to the computation. Agents have the necessary autonomy to make decisions, and interact with other agents and services to accomplish their goals. It is a new technology which can overcome many limitations of client/server architecture. The detailed knowledge information about mobile agent will be introduced in Section 3.1; *Active network* offers a technology where the application can not only determine protocol functions as necessary at the endpoints, but one in which applications can inject new protocols into the network or the network to execute on behalf of the application. The nodes of the network are programmable entities and application code is executed at these nodes to implement new services, the knowledge of active network is covered in Section 3.2.

3.1 Active Adaptation by Mobile Agent

Historically, distributed applications are created with "client/server" programming. In this model, an operation is split into two parts across a network, with the client making requests from a user machine to a server which services the requests on a large, centralized system. A protocol is agreed upon and both the client and server are programmed to implement it. A network connection is established between them and the protocol is carried out. The client/server model works well for certain applications. However it breaks down under lots of other situations, which include highly distributed systems, slow and/or poor quality network connections (such as wireline/wireless network), and especially in the face of changing applications.

With client/server architecture, it needs the following conditions to make good quality network connections [BWP98] :

- *First*, the client needs to connect reliably to its server, because only by setting up and maintaining the connection may it be authenticated and secure.
- *Second*, the client needs to be assured of a predictable response, since its many requests of the server require full round trips to be completed.
- *Third*, it needs good bandwidth, due to its very nature, client/server must copy data across the network.
- *Finally*, the protocol which a client and server agree upon is specialized and static. Often, specific procedures on the server are codified in the protocol and become a part of the interface. This interface is extensible, but only at the high cost of recoding the application, providing for protocol version compatibility, software upgrade, etc. As the applications grow and the needs increase, client/server programming rapidly becomes an impediment to change.

However, the conditions for client/server architecture to make good quality network connections cannot always be satisfied. We need more flexible technology, such as mobile agent technology, which can overcome the limitations of client/server. The following will introduce what is mobile agent, its advantages, and its applications related with active adaptations.

3.1.1 Mobile Agent and Its Advantages

Mobile agent, in simple words, is an independent software program running on behalf of a network user that can migrate from machine to machine in a heterogeneous network. The program chooses when and where to migrate. It can suspend its execution at an arbitrary point, jump to another machine and resume execution on the new machine [BKR98]. An agent may run when the user is disconnected from the network, even if the user is disconnected involuntarily. Some agents run on specialized servers, others run on standard platforms.

A mobile agent is specialized in that in addition to being an independent program executing on behalf of a network user, it can travel to multiple locations in the network.

As it travels, it performs work on behalf of the user, such as collecting information or delivering requests. This mobility greatly enhances the productivity of each computing element in the network and creates a uniquely powerful computing environment well suited to a number of tasks.

Mobile agents break the client/server barrier, and overcome all of the above inherent limitations in client/server:

- *Mobile agent shatters the very notion of client and server.* With mobile agents, the flow of control actually moves across the network, instead of using the *request/response* architecture of client/server. In effect, every node is a server in the agent network, and the agent (program) moves to the location where it may find the services it needs to run at each point in its execution.
- *The scaling of servers and connections then becomes a straightforward capacity issue, without the complicated exponential scaling required between multiple servers.* The relationship between users and servers is coded into each agent. It is the agent itself that creates the system, rather than the network or the system administrators. Server administration becomes a matter of simply managing systems and monitoring local load.
- *The problem of robust networks is greatly diminished.* The hold time for connections is reduced to only the time required to move the agent in or out of the machine. No requests flow across the connection, the agent itself moves only once, in effect carrying a greater "payload" for each traversal. This allows for efficiency and optimization at several levels.

Besides the above, mobile agents also have the following primary advantages [CHK97][BHN+97]:

1. *They facilitate high quality, high performance, economical mobile applications*: Applications employing mobile agents transparently use the network to accomplish their tasks, while taking full advantage of resources local to the many machines in the network. They process data at the data source, rather than fetching it remotely, allowing higher performance operation. They use the full spectrum of services

available at each point in the network and make best use of the network as they travel.

2. *Mobile agent technology provides for secure communications even over public networks*. Agents carry user credentials with them as they travel, and these credentials are authenticated during execution at every point in the network. Agents and their data are fully encrypted as they traverse the network. All this occurs with no programmer intervention.
3. *They efficiently and economically use low bandwidth, high latency, error prone communications channels.*
4. *They can offer dynamic adaptation:* Mobile agents have the ability to autonomously react to changes in their environment. However, such changes must be communicated to mobile agents from the mobile agent environment.
5. *They support for heterogeneous environments*: Both the computers and networks on which a mobile agent system is built are heterogeneous in character. As mobile agent systems are generally computer and network independent, they support transparent operation.
6. *They can personalize server behavior*: In the intelligent networks, mobile agents are proposed as a way to personalize the behavior of network entities (e.g., routers) by dynamically supplying new behavior.
7. *They are robust and fault tolerant:* The ability of mobile agents to react dynamically to adverse situations makes it easier to build fault tolerant behavior, especially in a highly distributed system.

We could also identify *four properties* of mobile agents that can be useful for the improvement of current network management, which are:

- **Intelligence** is the ability to not only perform the processing associated with a task, but to assume some level of control or decision making.
- **Autonomy** is the ability of the agent to operate independently, not merely in response
- **Cooperation** is the ability to interact with other agents. Cooperation encourages a hierarchical approach to problem solving where agents are assigned small low-level tasks and combine to achieve a higher level goal.

- **Mobility** is the ability to move the agent to a network component so that the agent operates in the same locale as the data. Agent mobility encompasses *code mobility*, which is transporting code to a network component for remote execution, as well as *migration*, which is the ability of a process to stop its execution, save its state and transport itself to another network component to continue its execution there. Both forms of mobility are powerful tools that lead to opportunities for service customization, software version control and upgrades, and more.

As mentioned above, mobile agent technology do have many advantages over the traditional client/server models. That is the reason we try to apply this technology to network management and distributed network systems.

3.1.2 Mobile Agent Technology in Distributed Multimedia Systems

In this section, we will explain on how mobile agent can be used in distributed multimedia systems.

3.1.2.1 Passive Approach

Presently, many distributed multimedia systems adapt to their changing environments and QoS requirements by exchanging control and feedback data between servers and clients. For the most part, the nature of such data is passive, that is, they contain values representing the states of some pre-defined variables and control parameters. The recipients of the data respond by executing a fixed set of functions, implementing some fixed adaptation algorithm.

The problem here is that these functions are indiscriminately applied to all participating machines even though the latter may have different requirements. For example, upon receiving feedback on frame loss from its clients, a video server reacts by executing the same frame dropping algorithm across all client, regardless of the difference in their processing powers. The only variability allowed is through changes in feedback values; the algorithm remains invariant.

In the above passive approach, all programs used to control QoS are static because they can neither be moved dynamically to a remote location nor replaced. They are attached to their hosts (server or client) for the entire duration of the application.

This absence of program mobility makes it very difficult for multimedia systems to support dynamic changes in QoS control policies and adaptation algorithms. Such changes are often required due to the time-dependent needs of an application which can best be served by switching to different protocols or different resource management policies. Accommodating these changes with a fixed set of programs would require a forecast of many possible scenarios, which may not always be attainable.

3.1.2.2 Mobile Agent To Rescue

In order to facilitate multimedia applications to timely adapt to their continuously changing environment and hence to QoS fluctuations, we can complement the passive data passing model with the active program passing model: *active* because programs can be dynamically launched, loaded and executed at a remote destination where computation is needed. Such programs are referred to as mobile programs or mobile agent. Dispatching a program to another computer is known as *remote delegation* [GY95] because the computational responsibility of the program is passed to another machine that carries out the actual execution.

Exploiting program mobility to actively adapt to applications' QoS variations offers some notable advantages [Tran97]:

- *A fine grained, dynamic customization of QoS control can be realized.*
  Through delegation, a client can inject into the server, at any point in time, a mobile program containing specific adaptation functions and algorithms, tailored to the client's processing capabilities and requirements. The program is then executed at the server, directly affecting the server's behavior towards the given client. This results in a flexible structure where different clients can impose their preferred QoS control policies carried in different mobile programs. For example, in multicast applications using MPEG 1 encoding, a group of clients may desire their server to adjust its frame

sending rates, while another group of slower clients may prefer the server to send the I frames and discard the rest.

- *Furthermore, existing QoS adaptation behavior can be altered dynamically to respond to QoS variations in a continuously changing environment.*

  Here, *re-delegation* can be used: a new program, which incorporates an application's new control intelligence reflecting new requirements, can be delegated over to a remote site, replacing the current mobile program. This strategy permits a video receiver to flexibly introduce new control schemes to its sender at any time during the application, thereby achieving far greater adaptivity than is traditionally possible. Considering a flow control example, if during a video play, a client's CPU is so busy that the current rate control strategy is deemed to be inappropriate, the client may decide to switch to a frame dropping strategy by simply re-delegating another suitable program.

The end result is that programs at the server no longer need to be fixed. Instead, a server can utilize dynamically deployed adaptation programs to respond to its clients' diverse hardware/software processing requirements. These programs can be removed and replaced on an 'as needed' basis when they become obsolete.

The mobility of programs can provide a richer set of adaptive capabilities to accommodate continuous QoS changes in distributed multimedia systems. The concept of active adaptation breaks the mindset of traditional adaptation. Instead of sending feedback values, clients have the latitude of dynamically sending feedback programs, which encapsulate their desired QoS control logic and data. These programs, customized for individual clients' needs, replace the server's rigid set of QoS control functions.

3.2 Towards Active Networks

Today, mobile programs are not only deployed in end-systems, but also in intermediary network nodes. This results in recent emergency of novel network designs and architectures, known as *active networks* [TW96]. The traditional view of a router/switch as being a passive 'store and forward' machine is replaced by a highly flexible engine, one

that can dynamically accept and perform customized computations on various packets according to individual application's requirements.

3.2.1 Programmable Network vs. Active Network

There has been an increasing demand to add new services to networks or to customize existing network services to match new application needs. However, the introduction of new services into existing networks is usually a manual, time consuming and costly process. The goal of programmable networking is to simplify the deployment of new network services leading to networks that explicitly support the process of service creation and deployment. Programmable network architectures can be customized by utilizing clearly defined open programmable interfaces (i.e., network APIs) and a range of service composition methodologies and toolkits.

A programmable network is distinguished from any other networking environment by the fact that it can be programmed from a minimal set of APIs from which one can ideally compose an infinite spectrum of higher level services [CKV+99]. The programmability of network services is achieved by introducing computation inside the network, beyond the extent of the computation performed in existing routers and switches. We can view the generalized model for programmable networks as comprising conventional communication, encompassing the transport, control and management planes, and computation as well. Collectively, the computation and communication models make up a programmable network.

Two schools of thought have emerged on how to make networks programmable: *Active Networks (AN)* [DARPA96] and *Open Signalling (Opensig)* [Open].

- The Opensig community argues that by modeling communication hardware using a set of open programmable network interfaces, open access to switches and routers can be provided; and by "opening up" the switches in this manner, the development of new and distinct architectures and services can be realized. Open signaling takes a telecommunications approach to the problem of making the network programmable. There is a clear distinction between transport, control and management that underpin programmable networks and an emphasis on service creation with QoS. The open

programmable network interfaces allow service providers to manipulate the states of the network using middleware toolkits (e.g., CORBA) in order to construct and manage new network services.

- The AN community advocates the dynamic deployment of new services at runtime mainly within the confines of existing IP networks. The level of dynamic runtime support for new services goes far beyond that proposed by the Opensig community, especially when one considers the dispatch, execution and forwarding of packets based on the notion of "smart packet". In active networks, code mobility represents the main vehicle for program delivery, control and service construction. Active networks allow the customization of network services at packet transport granularity, rather than through a programmable control plane. Active networks offer maximum flexibility in support of service creation but with the cost of adding more complexity to the programming model. AN approach is more dynamic than Opensig's network programming interfaces.

Both communities share the common goal to go beyond existing approaches and technologies for construction, deployment and management of new services in tele-communication networks. In this thesis, we put more importance on active network.

3.2.2 Introduction to Active Networks

As we know, traditional networks have the drawback that the intermediate nodes are closed systems whose functions are rigidly built into the embedded software. Therefore, development and deployment of new protocols in such networks requires a long standardization process. The range of services provided by the network is also limited because the network cannot anticipate and provision for all needs of all possible applications.

Active networks offer a different paradigm that enables programming intermediate nodes in the network. A network is active if it allows applications to inject customized programs into the network to modify the behavior of the network nodes [TW96]. This allows applications to customize the network processing and adapt it to the application's immediate requirements. This enables new protocols and new services to be introduced

into the network without the need for network-wide standardization. Active network suggests that protocols are nothing but services provided by the nodes of the active network. In an active network, applications have the ability to access these services and customize them for their needs.

3.2.3 Active Network Concepts

Traditional networking implementations follow a layered model that provides a well-defined protocol stack. Most implementations provide a fixed protocol stack that is determined by a long standardization process often taking many years and is fixed when the final system is constructed. Therefore the time delay from the conceptualization of a protocol to its actual deployment in the network is usually an extraordinarily long process.

Active networking offers a technology where the application can not only determine protocol functions as necessary at the endpoints, but one in which applications can inject new protocols into the network to execute on behalf of the application. The nodes of the network are programmable entities and application code is executed at these nodes to implement new services.

In this section, we are going to introduce two key concepts in active network: Smart Packet and active node.

3.2.3.1 Smart Packets

In an active network, data packets are information entities. These entities, which we call *Smart Packets* [KMH+98], contain a destination address, user data, and methods (or "how to" information) that are executed locally at any node in the active network. These methods turn the network elements into active elements: they apply the methods to the packets, thereby implementing network-based services tailored to the application. We can think of Smart Packets as carrying customized protocols that have to be fitted in with protocol modules at the network nodes.

The code in the Smart Packet can be in any executable format and it can be executed at the node if the node has the correct processing environment. Figure 3.1 and Figure 3.2 demonstrate the difference between *Smart Packets* and *Not-So-Smart Packets*.

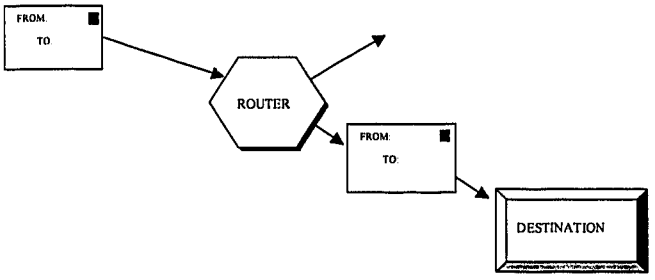


Figure 3.1 Not-So-Smart Packet
(Delivery Process is static, relatively passive)

3.2.3.2 Active Nodes

Nodes in an active network are called *active nodes* because they are programmable elements that allow applications to execute user-defined programs at the nodes.

Active nodes perform the functions of receiving, scheduling, executing, monitoring and forwarding Smart Packets [KMH+98]:

- When a Smart Packet arrives at an active node, the type identifier and the user-defined code inside the Smart Packet is extracted.
- The type identifier is used to de-multiplex the Smart Packet to its correct processing environment.
- The Smart Packet is then scheduled for execution.

A separate environment is required for each invocation to prevent undesirable interactions and malicious access to node resources.

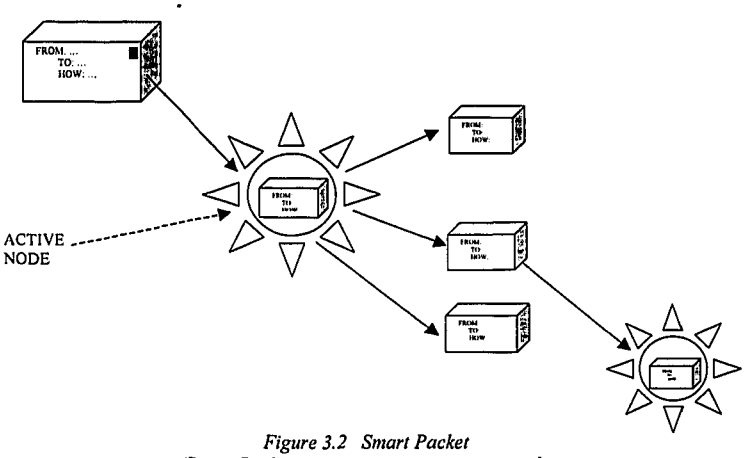


Figure 3.2 Smart Packet
(Smart Packet arrives, executes in active node, run, actively, then move on)

Active nodes export a set of resources and primitives that can be used by user programs. This not only provides a consistent view of the network but also enforces constraints on the actions that can be performed by user code.

3.2.4 Active Networks and Programming Interfaces

One way to think about active networks is they provide a *programmable* network API. As showed in Figure 3.1 and Figure 3.2, if we think of the IP header in the traditional network as the *input data* to a virtual machine, we can think of packets in the active network containing *programs* ("how to" information) as well as input data. In the context of this model, a variety of active networking approaches can be characterized by the following attributes [CBZ+98]:

- *Language Expressive Power*. The degree of programmability of the network API may range from a simple of fixed-size parameters that select from predefined sets of choices, to a Turing-complete language capable of describing any effective

computation. The advantage of a less powerful language is that it constrains the possible node behaviors and so simplifies correctness analysis.

- *Statefulness.* Another important characteristic of the network API is the ability to install *state* in the interior nodes of the network, and to refer to state installed by other packets. Some active network APIs provide this capability, while others do not. Where it is present, the API must include control mechanisms to protect users' state from unauthorized access.
- *Granularity of Control.* This refers to the scope of node behavior that can be modified by a received packet. One possibility is that a single packet can modify the node behavior seen by *all* packets arriving at the node, and this change persists until it is overridden. At the other extreme, a single packet modifies the behavior seen only by that one packet. Between these extremes, modifications might apply to a *flow*, which we define to a set of packets sharing some common characteristic, such as temporal locality and/or a particular source and destination address in the headers. In general, the active network API must include security mechanisms that ensure that packets affecting the node behavior have localized effect and/or come from authorized users.

The possibility of programming the network API introduces a new role, namely that of *service developer*: a third party who provides code that can be loaded into the active network to enhance or customize the service seen by users. Such code might be deployed by users themselves, or by network service providers.

3.2.5 Active Network Design Models

To help better understanding of active network, here we introduce some basic ideas about active network design models.

3.2.5.1 Design Space Axis

Two design space axis are important for designing an active network.

1. *The first axis addresses possible mechanisms for network programmability.*
   - At one extreme (called the "integrated approach" [TW96]), each packet (capsule) carries a program that may be evaluated at intermediate hops to effect its routing,

compute some useful result, or in some other way affect the network. Here, networking changes occur by changes at the programmable packet level.

- *Capsules – an integrated approach*

  Every message, or capsule, that passes between nodes contains a program fragment (of at least one instruction) that may include embedded data. When a capsule arrives at an active node, its contents are evaluated, in much the same way that a PostScript printer interprets the contents of each file that is sent to it. Bits arriving on incoming links are processed by a mechanism that identifies capsule boundaries, possibly using the framing mechanisms provided by traditional link layer protocols. The capsule's contents are then dispatched to a transient execution environment where they can safely be evaluated. The programs are composed of instructions, that perform basic computations on the capsule contents, and can also invoke "built-in" primitives, which may provide access to resources external to the transient environment. The execution of a capsule results in the scheduling of zero or more capsules for transmission on the outgoing links and may change the non-transient state of the node.

- The other extreme (called the "discrete approach"[TW96]) is that packets are passive, and that extensibility is provided by downloading code into the routers.
  - *Programmable routers (or switches) – a discrete approach*

    The processing of messages may be architecturally separated from the business of injecting programs into the node, with a separate mechanism for each function. This preserves the current distinction between *in-band* data transfer and *out-of-band* management channels. Users would first inject their custom processing routines into the required routers. Then they would send their packets through such "programmable" nodes much the way they do today. When a packet arrives at a node, its header is examined and the appropriate program is dispatched to operate on its contents.

    Separate mechanisms for loading and execution might be valuable when program loading must be carefully controlled. Allowing operators to dynamically load code into their routers would be useful for router

extensibility purposes, even if the programs do not perform application- or user-specific computations. In the internet, for example, program loading could be restricted to a router's operator who is furnished with a "back door" through which they can dynamically load code. This back door would at minimum authenticate the operator and might also perform extensive checks on the code that is being loaded.

- There is a mixture of these approaches in which packets carry programs that may refer to and invoke more general (and loadable) router-resident functionality.

2. *The second axis determines at which hops active evaluation should occur.*
   - The internet currently lies at one extreme: interesting 'active' processing can occur only at the endpoints.
   - Another view is that active processing should occur at every intermediate hop.
   - There is an intermediate position that allows evaluation at some of the intermediate hops, thus allowing more flexibility than end-to-end approaches while avoiding unnecessary processing overhead for simple tasks which do not require evaluation at every hop.

3.2.5.2 Towards a Common Programming Model

Network programs must be transmitted across the communication substrate and loaded into a range of platforms. This suggests the development of common models for: the encoding of network programs; the "built-in" primitives available at each node; and the description and allocation of node resources.

Program encoding. The objectives for program encoding are that they support:

- Mobility – the ability to transfer programs and execute them on a range of platforms.
- Safety – the ability to restrict the resources that programs can access.
- Efficiency – enabling the above without compromising network performance, at least in the most common cases.

Mobility may be achieved at several different levels of program representation:

- express the program in a high-level scripting language, e.g. Tcl;

- adopt a platform independent intermediate representation, typically a byte-coded virtual instruction set, e.g. Java;
- or transfer programs in binary formats, e.g., Omniware.

The Table 3.1 below describes recently developed enabling technologies that support the safe and efficient execution of each level of program encoding. All three approaches prove useful:

PROJECT	M	S	E	DESCRIPTION
Safe-Tcl (source)	X	X		Safe-Tcl (based on Tcl) is a scripting language that provides safety through interpretation of a source program and closure of its namespace. It depends on the restricted closure and correctness of the interpreter to prevent programs from deliberately or accidentally straying beyond their permitted execution environment.
Java (intermediate)	X	X	x	Java uses an intermediate instruction set to achieve mobility. Traditionally, the safe execution of intermediate code has relied on its careful interpretation. One of Java's key contributions is to improve efficiency by off-loading responsibility from the interpreter: the instruction set and its approved usage are designed to reduce operand validation per executed instruction.
Omniware (object-code)	x	X	X	Omniware portable object-code depends on software-based fault isolation (SFI) to enforce safety efficiently. It prescribes a set of rules that instruction sequences must adhere to, e.g. restrictions on how address arithmetic is performed. In conjunction with run-time support, these rules define a "sandbox" within which the program can do what it likes, but that it may not escape.
Proof-Carrying Code (object-code)		X	X	PCC uses a novel approach to achieve safety: it attaches a formal proof of the properties of a binary program. The recipient can check that the proof is valid, a process that is much simpler than constructing it from scratch. Currently, PCC is practical only short programs.

Table 3.1 Program Encoding Technologies (with labeled columns M, S, and E Assessing mobility, safety, and efficiency, respectively)

- source encodings support rapid prototyping;

- intermediate representations provide a compact and relatively efficient way to express short programs; and
- commonly used modules might best be expressed at the object-code level.

A possible approach to node interoperability would be to agree on an intermediate instruction encoding as the backstop for code mobility. Node implementers and users would be welcome to leverage alternative encodings, so long as they provide mechanisms through which an intermediate encoding of a program can be obtained or generated. Implementers may also leverage techniques such as dynamic ("on-the-fly") compilation that optimize common processing routines, both by converting portable representations to native ones, and by specializing programs to individual contexts. Operating system support for more specific strategies, such as "path"-based scheduling, protocol code reorganization, and low-level extensibility should also prove useful. The Table 3.2 below describes some of these compilation and operating systems technologies.

PROJECT	DESCRIPTION
Scout	Scout is designed to support communication-oriented tasks. It allocates and schedules resources on a "path" basis and applies a number of optimizations intended to increase throughput and decrease latency. Many of the techniques may be applicable to programs loaded into network nodes.
Exokernel	The exokernel enables programs to safely share low-level access to system resources. It implements a thin veneer that securely multiplexes the raw hardware. This in turn allows programs to tailor their own abstractions of operating system services, e.g., access to the active node environment.
SPIN	SPIN relies on the properties of the Modula 3 language and a trustworthy compiler to generate programs that will not stray beyond a restricted environment. Programs signed by the compiler may be dynamically loaded into the operating system.
'C	'C and VCODE enable "on-the-fly" code generation. This allows source programs to be automatically tailored, or even wholly generated, at runtime. These technologies could allow active nodes to translate commonly-used programs to binary encodings.

Table 3.2 Operating System Technologies

Common primitives. The services built-in to each node might include several categories of operations:

- primitives that allow the packet itself to be manipulated, e.g., by changing its header, payload, length, etc.;
- primitives that provide access to the node's environment, e.g., the node address, time-of-day, link status, etc.; and
- primitives for controlling packet flow, such as forwarding, copying, discarding.
- Additional primitives might provide access to node storage and scheduling, e.g., to facilitate rendezvous operations that combine processing across multiple packets.

Node resources and their allocation. Beyond encodings and primitives, there must be a common model of node resources and the means by which policies governing their allocations are communicated. The resources to be modeled include: physical resources, such as transmission bandwidth, processing capacity, and storage; as well as logical resources, such as routing tables and the node's management information base. Safe resource allocation is an area that will require considerable attention. Active nodes will be embedded within the shared network infrastructure, and so their designs must address a range of "sharing" issues that are often brushed over in the design of programmable systems destined for less public environments.

3.2.6 Brief Overview of Current Existing Active Network Technologies

In this section, we will introduce some active network technologies which were developed recently, which include PLANet [HMA+98] & SwitchWare [SFC+], ANTS [WGT96], NetScript [YS96], Smart Packet [SZJ+], etc.

3.2.6.1 PLANet & SwitchWare (University of Pennsylvania)

PLANet is an active network that is programmable in two ways. First, packets contain programs written in a special-purpose packet language called PLAN (Packet Language for Active Networks); these programs serve a role similar to the header of a traditional packet in providing control of how packets operate inside the network.

The SwitchWare project uses PLAN as the network API, it aims to build a software-programmable active network switch using a discrete approach. Programs could be dynamically loaded into the switch through the switch's input ports. Packets could then select between different network services by having different programs run on them.

3.2.6.2 ANTS (MIT)

ANTS (an Active Node Transfer System) was developed at MIT, which is an "active network" approach to building and deploying network protocols. This approach views the network as distributed programming system, and provides a programming language-like mode for expressing new protocols in terms of operations at nodes. It provides the greater flexibility that accompanies a programming language and the convenience of dynamic deployment.

The ANTS architecture has three key components:

- The packets found in traditional networks are replaced by *capsules* that refer to the processing to be performed on their behalf.
- Routers and end nodes are replaced by *active nodes* that execute capsule processing routines and maintain their associated state.
- A *code distribution mechanism* ensures that processing routines automatically and dynamically transferred to those nodes where they are needed.

Capsule

A capsule is a generalized replacement for a packet. Its most important architectural function is to include a reference to the forwarding routine to be used to process the capsule at each active node. Some forwarding routines are "well-known" in that they are guaranteed to be available at every active node, such as standard routing; Other routines are "application-specific", which will not reside at every node, but must be transferred to a node by the code distribution scheme before capsules of that type can be processed for the first time. Each capsule carries an identifier for its protocol and particular capsule type within that protocol.

Active Nodes

Active nodes execute protocols within a restricted environment that limits their access to shared resources. They exported a set of primitives for use by application-defined processing routines. They also supply the resources shared between protocols and enforce constraints on how these resources may be used as protocols are executed.

Code Distribution

The third component of ANTS is a code distribution system. Given a programmable infrastructure, a mechanism is needed for propagating program definitions to where they are needed. The ANTS couples the transfer of code with the transfer of data as an *in-band* function.

3.2.6.3 NetScript (Columbia University)

The NetScript project seeks to create a model for programmable, rather merely configurable, intermediate network node engine. NetScript uses agents to program management and control the functions of intermediate nodes. A NetScript agent glues primitive node functions to processing packet streams and allocate node resources. NetScript agents can be programmed to handle both standardized as well as non-standardized protocols. Packet streams arriving at intermediate nodes are processed by the appropriate agents to accomplish the desired functionality of these protocols.

The NetScript project consists of three components:

- *an architecture for programming networks in the large.*

  NetScript views a network as a collection of Virtual Network Engine (VNE) interconnected by Virtual Links (VL). The VNEs can be programmed by NetScript agents to process packet streams and relay these streams over VLs to other VNEs. The collection of VNEs and VLs defines a NetScript Virtual Network (NVN). NetScript provides a language to program a NVN. A NetScript program can be viewed as a collection of threads, distributed at the VNEs and processing packet streams moving through the NVN.
- *an architecture of a dynamically programmable networked device.*

- *a language called NetScript for building networked software on a programmable network.*

3.2.6.4 Smart Packets (BBN Technologies)

Smart Packets is an Active Networks project focusing on apply active networks technology to network management and monitoring without placing undue burden on the nodes in the network. Smart Packets improves the management of large complex networks by

- moving management decision points closer to the node being managed, targeting specific aspects of the node for information rather than scatter-shot collection, and
- abstracting the management concepts to language constructs, allowing nimble network control.

The Smart Packets architecture consists of four parts:

- a specification for smart packet formats and their encapsulation into some network data delivery service,
- the specification of a high level language, its assembly language, and compressed encoding representing that portion of a smart packet that gets executed,
- a virtual machine resident in each networking element to provide a context for executing the program within the smart packet, and
- a security architecture.

3.2.7 ANEP (Active Network Encapsulation Protocol)

One challenge in implementing Smart Packet is that IP does not have a notion of a datagram whose contents is processed at intermediate nodes. An IP router simply examines the datagram header and forwards the datagram. However, for Smart Packets in active network, the router must process the contents of the datagram before forwarding it. Further, the router should examine the contents of the datagram only if the router support Smart Packets. Otherwise, the router should pass the datagram through.

One solution is to modify an IP option, *Router Alert*, to achieve the operation specified above. The *Router Alert option* tells the router that it may need to examine the contents of

the datagram. Router Alert options can be specified for both IPv4 and IPv6. Based on the type tag, and possibly an examination of some of the higher-layer headers, the router can determine if it should process the datagram contents. If the router doesn't support active networks, it ignores the option and forwards the datagram. If the router supports Active Networks, it examines the ANEP (Active Network Encapsulation Protocol) message, learns the message is a Smart Packet and, if the router supports Smart Packet, it processes the packet.

This section specifies a mechanism for encapsulating active network frames for transmission over different media. This format allows use of an existing network infrastructure (such as IP or IPv6) or transmission over the link layer. This mechanism allows co-existence of different execution environments and proper demultiplexing of received packets.

An active network node (or active router) is capable of dynamically loading and executing programs, written in a variety of languages (such as PLAN, JAVA, etc.). These programs are carried in the payload of an active network frame. The program is executed by a receiving node in the environment specified by the ANEP. Various options can be specified in the ANEP header, such as authentication, confidentiality, or integrity.

Terminology

packet: an ANEP header plus the payload
active node: a network element that can evaluate active packets
TLV: acronym for Type/Length/Value constructs
basic header: the first two elements of the ANEP header

Reasons

The reasons an active network header is necessary are:

- An active node receiving a packet must be able to uniquely and quickly determine the environment in which it is intended to be evaluated.
- To allow minimal, default processing of packets for which the intended evaluation environment is unavailable.

- So that information that does not fit conceptually or pragmatically in the encapsulated program (such as security headers), can be placed in the header.

Packet Format

The packet format is shown in Figure 3.3.

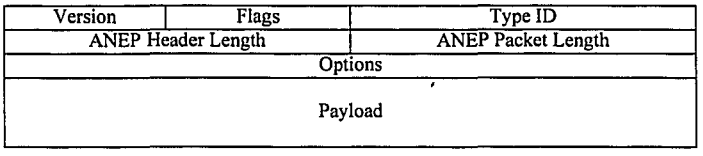


Figure 3.3 ANEP packet format

The *Version field* indicates the header format in use. This field will be changed if the ANEP header should change. If an active node receives a packet whose version number it does not recognize, it should discard the packet. The length of this field is 8 bits; The *flag field* is 8 bits long. It indicates what the node should do if it does not recognize the Type ID. If the value is 0, the node could try to forward the packet using the default routing mechanism. If the value is 1, the node should discard the packet; The *ANEP Header Length field* specifies the length of the ANEP header in 32 bit words. The length of this field is 16 bits; The *Type ID field* indicates the evaluation environment of the message. The active node should evaluate the packet in the proper environment. The length of this field is 16 bits. If the value contained in this field is not recognized, the node should check the value of the most significant bit of the Flags field in deciding how to handle the packet; The *ANEP Packet Length field* specifies the length of the entire packet; *Options* in the form of TLVs can included in the packet immediately following the basic header; The *Option Type field* identifies the option. How the active node handles the option Payload depends on the Option Type value.

Chapter 4
Existing Active Filter Applications

As the name suggests *active filters* are active technology based on mobile agent and active network. Active filters can be either nomadic or modifiable, or both. Active filters can be developed to be used in various areas, existing research on the applications of active filters includes: programmable congestion control; intelligent communication filtering; firewalls which are related to security issues; and e-commerce, such as online auctions, etc. This chapter is divided into two sections: Section 4.1 devotes to protocol classifications which are closely related to active filter functions in active network, such as filtering protocol class, combining class, transcoding class and network management class, etc. Existing active filter applications are introduced in Section 4.2, which include active networking and congestion control, intelligent communication filtering, firewalls, and online auctions, etc.

4.1 Protocol Classification in Active Networks

As we mentioned in Chapter 3, one of the biggest advantages of active network is that it has enabled us to develop and test new protocols within short time. While each protocol seems to be unique to an application, it generally possesses characteristics that are common to some other protocols. All such protocols can collectively be identified as members of a class. These include the architecture required for deployment, common protocol interfaces and the set of services required by the protocols from an active node. This facilitates rapid design of new protocols and the seamless introduction of these protocols into the network.

A few key issues to consider while developing the methodology for each protocol class are:

- *Architecture for deployment*: The architecture refers to the placement (location) of the protocol services in the nodes of the active network.

- *Common interface*: This means the structure of the Smart Packet and its design.
- *Node primitives required*: This describes the requirements of the protocol before it executes at an active node.

It is also important to find out if there is a *topology* for the distribution of a protocol inside the active network that is most efficient. For example, do we need to make all the nodes be active nodes in a network in order to implement active filtering? Or do we only need some critical network nodes to be active nodes?

We can identify the following protocol classes which are related to the functions of active filters:

4.1.1 Filtering Protocol Class

This encompasses all those protocols that perform packet dropping or employ some other kind of bandwidth reduction technique on an independent per-packet basis such as compression protocols and the transmission of layered MPEG: by prioritizing the layers, it is possible to maintain real-time connectivity in times of network congestion by dropping packets containing the lower priority layers.

Similar strategies are being used for audio transport wherein the signal components are separated based on their level of contribution to the original sound. Signal components that do not contribute heavily are placed in lower priority packets that are specially marked for discard if congestion occurs.

Protocols belonging to the filtering class are primarily developed to reduce bandwidth requirements of the application data. Temporary reduction in bandwidth requirements is necessary in the face of transient congestion problems. However, bandwidth reduction techniques are always required whenever there is a severe rate mismatch. This typically occurs at interfaces where there is an order of magnitude difference in the speeds on opposite sides of the interface e.g. the interface between wired and wireless networks. In such cases, it is obvious that the protocols have to be deployed at the interface gateway.

In an active network, applications could deploy the filtering code in this way: applications use *congestion detection module* (CDM) in the active nodes or just use Smart Packets to find out the occurring of a rate mismatch or congestion then installs (downloads) the filtering protocol at those active nodes. Since protocols of this class are designed primarily to reduce bandwidth requirements, the active node must supply them with the following primitives:

- To find out the available bandwidth over a particular interface.
- To find the maximum bandwidth capacity on all interfaces.
- Management of small state involves providing primitives for the creation of small state, and storage of information to and the retrieval of information from the small state.

4.1.2 Combining Class

The class of combining protocols has the property of combining packets that may come from the same stream or from different streams. For example, the Wireless ATM Voice/Data project [Wireless] combines two or more packets from the same stream to form a single packet that is forwarded to the next hop, the purpose of this technique is to reduce congestion; The Distributed Sensor Data Mixing [Yeadon96] project at Lancaster University is also a member of this class of protocols because it combines different streams into one: Different types of sensors such as microphones and antennas, dispersed over a wide area network, collect data and transmit them to receivers on the network. Instead of having each receiver do its own mixing of the transmitted data, some of the mixing is done within the network on the input signals that pass through the network node at approximately the same time. If the mixed signal is smaller than the sum of its constituents, then it reduces the bandwidth requirements and the processing to be done at the receiver.

Combining is an expensive processing step; therefore it is desirable for the active nodes deploying a protocol of this class to have sufficient processing power. They must also have sufficient memory storage because combining sometimes involves storing packets

from one stream until packets from the other arrive at the node. Therefore the interfaces that active nodes have to provide for this class of protocols are:

- Finding memory available for the Smart Packet.
- Management of small state.
- Primitives for cloning/duplication of Smart Packets to enable multicasting.

4.1.3 Transcoding Class

Protocols that transform the user data into another form within the network belong to the class of transcoding protocols. Examples of such protocols include encryption protocols and image conversion protocols. These protocols are CPU-intensive and therefore require nodes with sufficient computing resources. Encryption protocols are generally deployed only at the end-points of a connection whereas compression protocols are deployed either at the end-points or at points in the network where congestion likely happens and bandwidth control alternatives are desired. Protocols of this class are primarily processing functions and therefore the primitives desired are:

- Available memory.
- Computing resources.

4.1.4 Network Management Class

The programmability of the nodes in an active network enables the creation of self-configuring, self-diagnosing and self-healing networks. This involves actions such as alarm and event reporting and workload monitoring, etc. The advantage of using active network to perform such functions is that it is possible to capture a consistent state of a node by sending a single Smart Packet that gathers all relevant information at one time.

The architecture involved in the deployment of these protocols requires dynamically establishing monitoring and monitored entities in network. Alarm and event reporting functions have to be defined and installed at various nodes in the network. The active nodes must provide management Smart Packets with the following interfaces:

- To create, access and modify the state of the active node.
- To establish events and state for which information is to be gathered.

- To establish frequency and format of reporting event information.

As a generalization, active filters may perform the functions of filtering, combining, transcoding, network management (such as rerouting based on real-time network conditions).

4.2 Existing Active Filter Applications

In this section, we will introduce the existing active filter applications, which includes: active networking and congestion control; intelligent communication filtering; firewalls; online auctions, etc.

4.2.1 Active Networking and Congestion Control

Active Networking (AN) [BCZ97] refers to the addition of user-controllable computing capabilities to data networks. With active networking, the network is no longer viewed as a passive mover of bits, but rather as a more general computation engine: information injected into the network may be modified, stored, or redirected as it is being transported.

In this approach, users can select from an available set of functions to be computed on their data, and can supply parameters as input to those computations. The available functions are chosen and implemented by the network service provider, and support specific services; thus users are able to influence the computation in a way of choosing from a selected functions, but cannot define arbitrary functions to be computed.

4.2.1.1 An Architecture for Active Networking

A Generic Model of Packet Processing

The network consists of switching nodes, which are connected via links. In this generic model, nodes don't do anything except process the packets received on their incoming links; processing an incoming packet may result in one or more packets being transmitted on outgoing links. More precisely, the state of a node comprises the following pieces:

- An input queue of packets. Packets received on any link are placed in the input queue.

- For each outgoing link, an output queue containing packets to be transmitted on that link.
- A collection of generic state information. This represents long-lived information maintained at the node, such as routing tables or virtual-circuit switching tables.

Each node in the network supports a particular set of functions, each of which has a unique identifier. Each packet contains a set of headers, which specify (i) the identifier of one or more functions to be applied to the packet; and (ii) parameters to be supplied to those functions. When the packet is processed, the function identified by each header is applied, resulting in updating of the node's state and possibly modification of the rest of the packet.

For each function identifier *f*, and each parameter value *p* for function *f*, there is a particular subset of the node's generic state information that is relevant to *f* and parameter *p*. Functions cannot modify or use parts of the node state that are not relevant.

Each node repeatedly performs the following:

```
Remove a packet M from the input queue;
while (more functions need to be applied to M):
   Let f, p be the function ID and parameter from the next header of M;
   Let g be the state component relevant to f and p;
   Invoke function f on M, with p as parameter:
      (optionally) Modify M;
      (optionally) Update g;
      (optionally) Queue messages for output;
```

Traditional networking functions can be characterized as node-processing functions in this model.

From Packet Forwarding to Active Networking

This approach defines active networking to be extension of the set of functions that can be invoked at a network node beyond those required to simply move bits from place to place. The basic idea of active networking is the incremental addition of user-controllable functions, where each function is precisely defined and supports a specific service.

In general, the introduction of new AN functions involves specification of the following:

- The *identifier* associated with the function.
- The *parameters* associated with the function, and the method of encoding them in a packet.
- The *semantics of the function.* A standard environment, comprising support services such as private state storage and retrieval, access to shared state information (e.g. routing tables), message forwarding primitives, etc., would provide a foundation on which new AN functions services could be built.

In the view of AN, addition of a new function to a network node would be the responsibility of the network service provider. This approach corresponds roughly to the way new features are deployed in the public switched telephone network today: users have the option of provisioning various features implemented and deployed by the service provider.

4.2.1.2 Programmable Congestion Control

Operating Model

From the point of view of a node somewhere in the network, a flow is a sequence of packets all having the same source and destination. A flow might consist of packets traveling between a single pair of end-points, or it might be the aggregation of a set of lower-level flows. It is assumed that a flow is identified by a label of some kind in the network protocol header.

Generally, programmable congestion control operates as follows: Based on triggers that indicate congestion control should take place, flow state is examined for advice about how to reduce quantity of data. The important components of this model are:

- the *triggers* responsible for initiating congestion control,
- the *flow state* that contains the specific advice for this flow, and
- the *reduction techniques* defined by the network and made available to the users.

An important feature of this model is its consistency with traditional best-effort service. That is, a flow provides advice about what to do with its data. The network node is not required to take the advice, and may apply generic bandwidth reduction techniques.

This approach focus on the special case of *intelligent discard of data.* It allows applications to define units based on application semantics, with aim of discarding the entire unit if any portion must be discarded. Given that bandwidth reduction will occur by discarding units, a question arises as to which units (within a flow) to discard:

- In the most simple case, there is no choice: when the congestion indication trigger occurs, a fixed unit (typically the one currently being processed) is subject to discard.
- More efficient network behavior is possible. The congestion control advice was considered, which indicating priority or some other policy by which to discriminate across data in the same flow.

Making use of this advice clearly requires that the network node have access to a collection of data within a single flow. These mechanisms involve storing and manipulating flow data before it leaves the node, e.g., while sitting in a per-flow queue from which packets are periodically selected for output by a scheduling mechanism.

4.2.1.3 Application and Mechanisms of Congestion Control to MPEG

As we introduced in Chapter 2, Section 2.5, the important feature of an MPEG stream is that it consists of a sequence of frame of three types: I-frame, P-frame, B-frame. Coding dependencies exist between the frames, causing P- and B-frames to possibly require other frames in order to be properly decoded. Each I-frame plus the following P- and B- frames forms a group of pictures (GOP), which can be decoded independently of the other frames.

The specific components of the programmable congestion control are implemented as follows:

- *Source-attached advice*. The mechanisms were considered, in which the source identifies "units" such that the unit will be discarded if any portion of the unit must be dropped.
- *Frame Level Discard* mechanism defines a unit to be an MPEG frame. The advice given is to queue a datagram if and only if its corresponding frame can be queued in its entirety. The state for each frame was maintained that is being discarded or buffered, and use this state information to decide, in constant time, to buffer or discard a particular datagram.
- A mechanism that identifies dependencies between units is further considered. *Group of Picture (GOP) Level Discard* maintains state about the type of frame discarded. In case an I frame has been discarded, the corresponding P and B frames are discarded as well.
- *Choice among units*. A policy for making choices amongst units is considered. When a I frame is too large to be accommodated in the output queue, and the queue contains P and B frames such that their combined sizes are greater than that of the I frame, then such P, B frames are discarded, and the I frame transmitted.
- *Triggers*. Two types of triggers are considered. In the first, This approach detect and respond to congestion only when data arrives that cannot fit in the output queue. All three mechanisms mentioned above use this trigger. An "early" trigger, which detects and responds to congestion when the output queue occupancy exceeds a certain threshold are also being considered.

4.2.1.4 Limitations

This approach has some benefits with respect to incremental deployment as well as security and efficiency: AN functions can be individually implemented and thoroughly tested by the service provider before deployment, and new functions can be added as they are developed. However, there are some tradeoffs. As we can see, users can only select from an available set of functions to be computed on their data. The available functions are chosen and implemented by the network service provider; thus users can only be able to influence the computation of a selected function, but cannot define arbitrary functions to be computed.

4.2.2 Intelligent Communication Filtering

A mobile computer may move through areas that provide wide variety of operating conditions. In particular, it may be attached to a high speed (wired) network at one moment and to a low speed, pay-per-use (wireless) network at the network moment. Most distributed systems can be expected to react poorly to such sudden, drastic changes in available bandwidth.

A new architecture for distributed systems supporting mobility was advocated [ZD94]. In this architecture an intermediary filter is interposed between client and server. Its purpose is to filter or delay all but the most essential data that would travel over the slow link to the mobile host.

The actions of the intermediary filter are controlled by the client, since the client is more likely to be informed of the circumstances that motivate data filtering, and since the network link to the client is probably the "cause of the problem." Types of actions that the intermediary might take include:

- Running an optimized version of a protocol between itself and the mobile host.
- Omitting data or reducing interactions.
- Delaying transmission of some data, forcing the client to demand-fetch it.
- Compressing data.
- Compression and decompression are properly placed at the intermediary and mobile host, rather than "end-to-end" because the need for compression arises from link characteristics.

The filters were expected to embody a significant amount of application specific knowledge, and would have to accumulate state in order to make effective filtering decisions.

A number of advantages derive from filtering data at the intermediary filters:

- Since the communication link between intermediary and mobile host may be slow, reducing the amount of traffic can improve performance.

- Even if data filtering reduces performance it may reduce cost. Depending on the relative degrees of reduction, reduced cost in return for reduced performance might be desirable.
- If properly designed, the intermediary can act not only as a filter, but also as an "agent": i.e., as a permanent representative for a mobile host that might not always be powered up or connected to the Internet.

Design of Intelligent Communication Filter

The following will introduce how to interpose an intermediary, add a filter and associate a filter with a stream.

(1) Interposing an Intermediary

The intermediary is realized as a process called the *Proxy Server*. The Proxy Server runs on some host, presumably in the wired part of the network. It could be advantageous to run the Proxy Server on a host that is the fringe and hence attached to both wired and wireless links. Such a host might have extra information about the characteristics of the wireless link. However, there is no constraint on where the Proxy Server may run.

The Proxy Server handles all traffic between the mobile host and the outside world, which includes both filtered and unfiltered data.

(2) Adding a Filter

A filter is hard-coded into the Proxy Server and automatically attached to data streams created by the mobile host.

(3) Associating a Filter with a Stream

When the mobile host starts a process, a "session" is created with the Proxy Server which lasts the lifetime of the process. Within the proxy server, two threads are created to handle data coming from and going to the mobile host's process, respectively. Such threads call a filter function, handling packets to it and possibly receiving packets back.

Unresolved Issues

There is no design of a programming interface through which arbitrary filter code can be dynamically loaded into the Proxy Server and attached to an arbitrarily defined data stream. The filter could be written in a simple interpreted language containing primitive actions (forward, discard, etc.) to be executed based on the contents of the packet.

Where to place the Proxy and when to move it is an issue that remains to be investigated. It seems desirable to have the Proxy located near the mobile host it is serving; however, it certainly is undesirable to move the Proxy too often in response to a highly mobile host.

4.2.3 Firewalls

Firewalls implement filters that determine which packets should be passed transparently and which should be blocked. Although they have a peer relationship to other routers, they implement application- and user-specific functions in addition to packet routing. The need to update the firewall to enable the use of new protocols is an impediment to their adoption. In an active network, this process could be automated by allowing applications from approved vendors to authenticate themselves to the firewall and inject the appropriate modules into it.

4.2.4 Online Auctions

Web servers hosting online auctions are currently among the most popular sites in the Internet. A server running a live online auction collects and processes client bids for the available item(s). This server also responds bids for the current price of an item. Because of the network delay experienced by a packet responding to such a query, its information may be out of date by the time it reaches a client, possibly causing the client to submit a bid that is too low to beat the current going price. Thus, unlike auctioneers in traditional auctions, the auction server may receive bids that are too low and must be rejected, especially during periods of high load when there are many concurrent bids.

Current implementations of such servers perform all bid processing at the server. In an active network, low bids can be filtered out in the network, before they reach the server. This capability can help the server achieve high throughput during periods of heavy load.

When the server senses it is heavily loaded, it can activate filters in nearby network nodes and periodically update them with the current price of the popular item. The filtering active nodes drop bids lower than this price and send bid rejection notices to the appropriate clients. This frees up server resources for processing competitive bids and reduces network utilization near the server. The filtering active nodes could also keep track of the number of rejected bids at each price, and ship those to the auction server at the end of the auction. The auction server performs caching (of current price information) in network nodes.

The essential feature of the auction service is that low bids may be rejected at nodes within the network when server load is high. The basic form of this functionality can be realized can realized in ANTS (an active network architecture developed at MIT which we introduced in 3.2.5.2) with a protocol comprised of four capsules:

- a FILTER capsule for the server to set a filtering price.
- a BID capsule for clients to submit bids.
- a SUCEED capsule for the server to notify a client that a bid succeeded.
- a FAIL capsule to notify a client that a bid failed or would have failed.

During the normal operation, BID capsules are sent from clients to the server, and SUCCEED and FAIL capsules returned from the server to client. Unlike traditional auctions, bids may fail to be accepted because they are out-of-date by the time they are processed at the server. During periods of high load, many bids may fail, and the server may delegate some rejection processing to active network node. It does this by sending FILTER capsules to nearby active nodes. These capsules store the current price in the node, and subsequent BID capsule passing through the node compare the price of their bid with a known bid. If it is lower, then a FAIL capsule may be returned from within the network indicating failure, and the BID capsule need not be forwarded to the server. Note that the SUCCEED capsule is generated only by the server, never by interior network nodes.

The FILTER capsule uses a flooding algorithm to update the current price of the item at all network nodes within a certain radius of the server; the size of the radius in hops is selected by the server depending on load. At each load it reaches, it updates the item's price in the cache, decrements its own hop limit, and then forwards copies of itself on all outgoing links. Forwarding stops when the hop limit is exhausted, or if it reaches node that has filter that supersedes the one being forwarded. The BID capsule forwards itself towards the server, comparing its bids with any known prices it discovers along the way. Strictly lower bids are rejected by creating a FAIL capsule and returning it to the sender in place of forwarding the failed BID. The processing routines for the FAIL and SUCCEED capsules are not shown, since these capsules are simply forwarded at nodes until they reach their destinations.

Chapter 5
Extended Use of Active Filters in Other Domains

In this chapter, according to the filtering theories introduced in Chapter 2, by using the mobile agent and active network technologies which were discussed in Chapter 3, and based on Chapter 4's existing active filter applications, we extended and developed the active filter functions into other domains, such as: real-time resource management; rerouting; application-specific filtering in wireline/wireless network; and real-time multicast. Therefore, this chapter is divided into four sections: Section 5.1 covers active filters in real-time resource management, which includes the motivation of our research and design of an active filter architecture. Section 5.2 devotes to load-sensitive rerouting via active filters. Active filters could also be applied to wireline/wireless network, which is discussed in Section 5.3; In Section 5.4, we designed another active filter architecture which can be used to improve the efficiency of real-time multicast.

5.1 Active Filters in Real-time Resource Management

In the design of active filter in real-time resource management, active filter is a code segment that applications or service providers inject into the network to assist in the runtime management of the network resources that are allocated to them.

This active filter architecture is driven by two requirements. First, users should be able to tailor runtime resource management so that they can optimize their notion of quality of service. Second, since active filters execute inside the network, they can quickly respond to changes in the network conditions. We also describe the programming interface that active filters can use to monitor the network conditions, e.g. queue status and bandwidth of the flows they are responsible for, and to modify resource use, such as selective packet dropping, rerouting, and changing reservations, etc.

Active filter is a mechanism (code segments) for applications and service providers to inject into the network that are directly involved in or affect the resource management decisions for the traffic belonging the that user. Service providers must be able to influence how "their" resources are managed based on their own notion of service quality, and this is most directly achieved by having them provide code (active filter) that implements their policies. We also call these code segments *delegates* since they represent the interests of the users inside the network. These delegates can be developed to handle problems such as congestion control for video streaming and balancing traffic load.

5.1.1 Motivation

Advanced applications will have many flows that use a variety of resources in the network. Runtime resource management policies are needed in a number of situations:

- First, the availability of networking resources may change, forcing the application to change how it uses resources; this is most important for applications that use best effort service or weak guarantees.
- Alternatively, the application may have to change its resource usage because its requirements have changed.

As we have already known, responsibility for adapting resource use has traditionally been pushed to the end points, this simplifies the core of the network. However, both network applications and the network itself are changing rapidly. Applications are becoming more complex and sophisticated. The network provides mechanisms for explicit resource control and is delivering more sophisticated services. As result of these changes, having some resource management policies implemented by entities inside the network could have several advantages:

- *Strategically placed entities in the network can more easily collect all the information that is needed to make resource management decisions.* For example, they could monitor how all flows belonging to a user are using a congested link. An endpoint typically has information only on the flows it generates or receives.

- *Entities in the network have immediate access to relevant information and can more quickly respond to changes.* Adaptation policies implemented at end-points have to deal with at least one round-trip time worth of delay.
- *Endpoints of course have to be involved in runtime adaptation. However, entities in the network can give specific feedback that may help in adapting.* Without explicit feedback endpoints have to rely purely on implicit feedback, i.e. packet loss or measured delay; implicit feedback is often hard to interpret and often offers incomplete information.

Motivated by these potential benefits, we feel that it is necessary to implement an application-specific or service-specific runtime resource management policies inside the network by using mobile agent and active network technologies called active filters.

5.1.2 Active Filter Architecture

In this part, we describe the active filter runtime environment, focusing on the programming interface that active filters use to perform customized runtime resource management. Since active filters use this interface to control the router's behavior, we call it *Router Control Interface (RCI)*.

The most directive way of having applications and service providers involved in runtime resource management is to have them provide code that implements their adaptation policies. Applications or service providers can inject active filters into the network to implement customized resource management of their data flows.

We divide active filters into *active data filters* and *active control filters*. Active data filters can be used to implement data manipulation operations such as video transcoding, compression, or encryption, they need large memory and computing resources. Active control filters, on the other hand, perform resource management tasks that do not require processing or even looking at the body of packets, such as changing bandwidth allocations, selective packet dropping, or rerouting. Active control filters could execute on the control processor of routers or switches.

Active filters execute on designated active routers and can monitor the network status and affect resource management on those routers. The network model that forms the basis for the router control interface that active filters use is illustrated in Figure 5.1. The traffic in the network is viewed as a set of flows (a sequence of packets with a semantic relationship defined by application and service providers). Flows are defined on each router using a *flow spec*, i.e. a list of constraints that fields in the packet header must match for that packet to belong to the flow. A *packet classifier* in the *data plane* of the router (shown in white in Figure 5.1) determines what flow each incoming packet belongs to (Figure 5.2). The active control filters live in the *control plane* (shown in gray in Figure 5.1) of active router and can monitor and change resource use in the data plane on a per-flow basis.

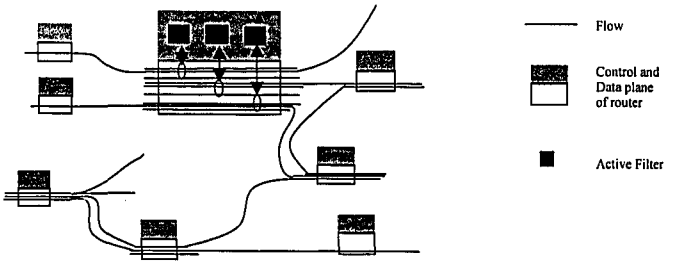


Figure 5.1 Active Filter Network Model

The distinction between control and data filters is in part driven by the desire to achieve good performance using today's routers. Complex data manipulation operations like data filters could be moved to computer servers, so that the router data plane can remain simple: it only has to perform classification and scheduling. In contrast, there is more room in the control plane for customization and intelligent decision making using active control filters. However, even on different router architectures, e.g., routers that can

support expensive data manipulation, the distinction will be useful, since the two types of filters need different RCIs, and raise different performance and security concerns.

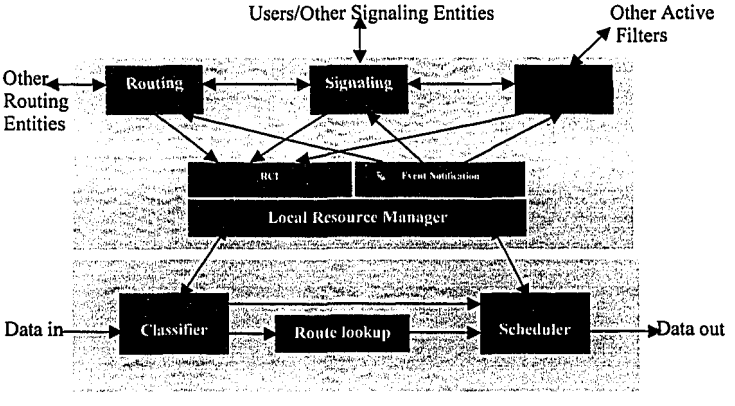


Figure 5.2 Active node architecture (Adapted from [CFK+98])

A critical design decision for active filters is the definition of the router control interface, i.e., the RCI that active filters use to interact with the environment. If the RCI is too restrictive, it will limit the usefulness of active filters, while too much freedom can make the system less efficient.

The definition of the RCI is driven by the need to support resource management and it includes functions in three categories:

- **Collecting information**: Active filters can monitor network status, waiting for events such as congestion conditions or hardware failures, or just keeping track of traffic patterns and flow distributions. Querying output queue sizes, checking for connectivity, or retrieving bandwidth usage are methods that can be used to collect information local to an active filter.
- **Resource management actions**: Active filters can change how resources are distributed across flows: splitting and merging flows, changing their resource

allocation and sharing rules. For instance, a subset of a flow may be isolated through a flow split, and assigning no resources to that subset implements a selective packet dropping mechanism. Active filters can also affect routing, for example to reroute a flow inside the application's traffic for load balancing reasons. Another example is to direct a flow to an active data filter on a compute server that will, for example, perform data compression to reduce bandwidth usage.

- **Active filter communication:** Active filters can send and receive messages to coordinate activities with peers on other routers and to interact with the application on endpoints. Messaging between active filters allows the global knowledge and perform global actions, as in the case of rerouting for load balancing. Interaction with end-points increases the flexibility of the system, as adaptation to network events typically involves the sources.

5.1.3 Active Filter Runtime Environment

Our current framework for active filters is based on Java and uses the Java virtual machine, capable of just-in-time (JIT) compilation and available for many platforms. We hope that this environment will give us acceptable performance, portability, and safety features inherited from the language. Active filters can be executed as Java threads inside the virtual machine "sandbox." Table 5.1 presents the methods that implement the RCI

Methods	Description
add	Add node in scheduler hierarchy
del	Delete node frm scheduler hierarchy
set	Change parameter on scheduler queue
dsc_on	Activate selective discard in classifier
dsc_off	Deactivate selective discard in classifier
probe	Read scheduler queue state
reqMonitor	Request async. cong. notification
retrieve	Retrieve scheduler queue state
getrt	Get next hop's IP address for a specified destination
chgrt	Change the routing table entry for a specified destination
mmode_on	Turn on the monitor mode to monitor bandwidth and delay
mmode_off	Turn off the monitor mode
getdata	Retrieve bandwidth usage and delay data recorded in the kernel

Table 5.1 RCI calls available to the active filters

to the packet classifier, scheduler and router. Communication can be built on top of standard *java.net* classes. While this environment is sufficient for experimentation, it is not complete. It needs support for authentication and mechanisms to monitor and limit the amount of resources used by active filters.

5.1.4 Active Filter Set Up

Setting up an active filter involves a number of steps:

- First, we have to verify that the router is an active router which has sufficient CPU and memory resources to support the active filter. The active filter may also need specific libraries or APIs that may or may not be available on normal routers. Verifying that these conditions are met is a form of admission control.
- Second, the active filter code has to be transferred to the active router and installed.
- Finally, the active filter runtime environment has to be told what flows it is responsible for.

Active filters are characterized by their QoS requirements, runtime environment needed (e.g., Java, Perl, VisualBasic script, etc.). Runtime type identifies the native library requirement of the active filter (e.g., JDK 1.0.2, WinSock 2.1, etc.). In addition to active filter QoS and runtime requirements, the active filter setup message also contains a list of flow descriptors, which identify flows to be manipulated by the active filter at the execution active node. At the execution active node, when an active filter setup message arrives, the appropriate runtime environment is located, the active filter is instantiated and then is passed to the local resource manager. By using these handles, the active filters can interact directly with the local resource manager to perform resource management for the flows during runtime.

5.1.5 Implementation

This part briefly describe the conceptual testbed, and present how active filters can be used to perform customized runtime resource management.

A. Conceptual Testbed

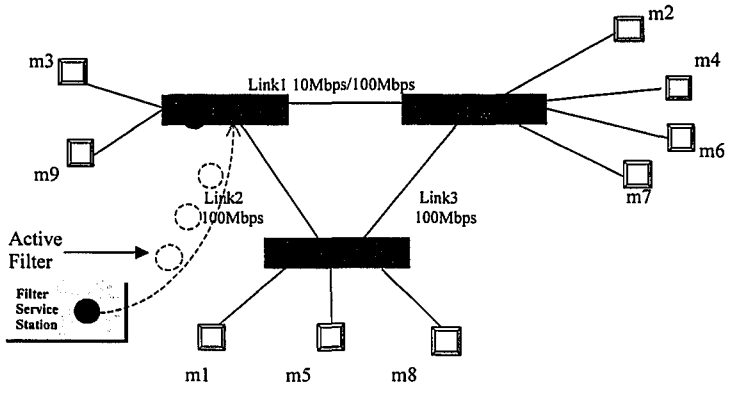


Figure 5.3 Testbed Topology

The system will be implemented on a testbed of PCs. The topology of the conceptual testbed is shown in Figure 5.3. The three routers can be Pentium II 266 MHz PCs. The end systems `m1` through `m9` could be workstations running Unix 4.0. All links are full-duplex point-to-point Ethernet links configured as 100 Mbps.

B. Selective packet dropping for MPEG video streams

As we mentioned in Chapter 2, Section 2.5, MPEG video streams are very sensitive to random packet loss because of dependencies between three different frame types: I frames (intracoded) are self contained. P frames (predictive) uses a previous I or P frame for motion compensation and thus depend on this previous frame, and B frames (bidirectional-predictive) use (and thus depend on) previous and subsequent I or P frames. Because of these inter-frame dependencies, losing I frames is extremely damaging, while B frames are the least critical. In this section, we will show how active filters can be used to selectively protect the most critical frames during congestion.

To create congestion, we can direct three flows over the PC 1 - PC 2 link of the testbed: two MPEG video streams and one unconstrained UDP stream. Both video sources send at a rate of 30 frames/second, the performance metric could be defined as the rate of correctly received frames. We can then compare the performance of the following four scenarios.

- In the first scenario, the video and data packets are treated the same, and the random packet losses should result in a very low frame rate.
- In the second case, the video stream share a bandwidth reservation equal to the sum of the average video bandwidths. This should improve performance. But the video streams are bursty, and random packet loss during peak transfers still results in large amount of frames cannot be received correctly.
- In the third scenario, we place an active filter on PC 1. The active filter monitors the length of queue used by video streams using the *probe* call. If the queue grows beyond a threshold, it instructs the packet classifier to identify and drop B frames. This is done by setting up the B frames as a separate flow using the *add* call (B frames are marked with an application-specific identifier), and then switching on selective discard for that flow using the *dsc_on* call. Packet dropping is switched off when the queue size drops below a second threshold.

  While active filters provide an elegant way of selectively dropping B frames, the same effect could be achieved by associating different priorities with different frame types.
- In scenario four we can use an active filter to implement a more sophisticated customized drop policy. In scenario three, either all or none of the B frames are dropped. By dropping the B frames of only a subset of the video streams, we can achieve finer grain congestion control. To achieve this, we can use a simple "time sharing" policy, where every few seconds the active filter switches the stream that has B frames dropped. This should further improve the performance.

C. Dynamic control of MJPEG video encoding

An alternative to selective frame dropping for dealing with congestion is to use a video transcoder to compress, or change the level of compression, of the video stream. It is still

possible to dynamically optimize video quality, as in the previous example, by using an active filter to control the level of compression.

In this experiment, we design an application consisting of two MJPEG video streams and two bursty data streams is competing for network bandwidth with other users, modeled as an unconstrained UDP stream. All flows are directed over the 10 Mbps PC 1- PC 2 link. The application has 70% of the bandwidth, 20% for video and 50% for data, and remaining 30% is for the competing users. The application data streams belong to a distributed FFT (fast Fourier transforms) computation. Since FFT alternates between compute phases, when there is no communication, and communication phases, when the nodes exchange large data sets, the data traffic is very bursty. In this experiment, the video flows have priority on taking bandwidth not used by the FFT flows. This means that video quality can be improved significantly during the compute phases of the FFT, if the video can make use the additional bandwidth.

This can be achieved by having an active filter on PC 1 monitor the FFT traffic, and adjust the level of compression of a transcoder (an active data filter) executing on the server `m9`. The transcoder takes in raw video and generates MJPEG. This allows the video flows to opportunistically take advantage of available bandwidth.

D. Selective dropping of non-adaptive flows

Applications that do not use appropriate end-to-end congestion control are an increasing problem in the Internet. These applications do not back off when there is congestion, or they back off less aggressively than users that use correct TCP implementations, and as a result, they get an unfair share of the network bandwidth. Such flows are called as *non-conformant* flows. In response to this problem, researchers have developed a variety of mechanisms that try to protect conformant flows from non-conformant flows. These include Fair Queuing scheduling strategies that try to distribute bandwidth equally, and algorithms such as RED [FJ93] and FRED [LM97] that, in case of congestion, try to selectively drop the packets of non-conformant flows.

Once deployed, these mechanisms will improve the fairness of bandwidth distribution at the bottleneck link, however, they address only part of the problem since they are designed to work locally. The problem is that non-conformant flows still consume (and probably waste) bandwidth upstream from the congested link. Upstream routers may not respond to the non-conformant flows, for example because they have no support for detecting non-conformant flows, or because the flow cannot be detected, or because the flow appears to be conformant (e.g., does not cause congestion). This problem can be addressed by having routers propagate information on the non-conformant flows upstream along the path of those flows.

We can implement a simple version of this solution using active filters. An active filter locally monitors the congestion status and tries to identify non-conformant flows among the flows it is responsible for. Once a "bad" flow has been identified (in the implementation, a flow can be considered to be non-conformant flows if its queue is overflowing for an extended period of time), the active filter enables selective packet dropping for the flow, and sends the flow's descriptor to a peer active filter on the upstream router. When an active filter receives a report of a "bad" flow, it verifies that the flow indeed has a high bandwidth and enables selective packet dropping, if possible, and forwards the message to the upstream router. Clearly, many alternative policies could be implemented, for example, only a certain percentage of the packets could be dropped to reduce its bandwidth instead of dropping all packets as in our experiment design.

5.2 Load-sensitive flow rerouting via active filters

In a telecommunications network, a call between two parties may be connected via one of a number of paths. The process of deciding which of these paths to use is called *routing*. Choosing an efficient path is important because the network's capacity for handling traffic is finite. However, finding the optimal path is problematic because the network state continually evolves. By the time the information needed to compute the optimal path between any two nodes is made available at the node where that decision needs to be taken, the network state will probably have changed, rendering that decision obsolete. Furthermore, efficient routing decisions, those which maintain a balance in utilization of

the network resources, require information about the utilization of *all* network resources to be made simultaneously available to the process making that decision.

Routing algorithms are used to establish the appropriate routing paths or the equivalent routing table entries in each node along a path. Most algorithms are based on assigning a cost measure to each link in the network and determining the linear sum of paths across the network. Based on these costs, the network tries to allocate traffic to the cheapest paths across the network. Where the cost function is based on link congestion (real or predicted) the cheapest path may change over time to maintain the network level efficiency. However, such mechanisms are limited by their lack of information about the wider network state, which means that the traditional routing approaches cannot determine the most efficient path from the network point of view merely by checking a small number of paths for congestion.

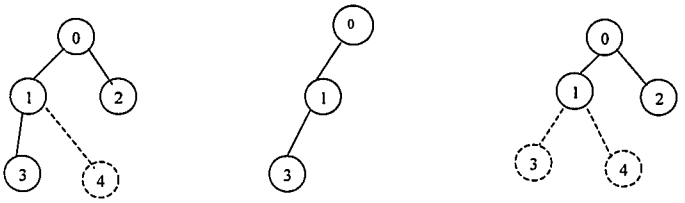
Routing decisions in Internet today are mostly load-insensitive and application-independent; in other words, the path taken by a packet does not depend on the load in the network or the application the packet belongs to. While this results in simple and stable routing protocols, it can also cause inefficient use of network resources. For example, in a client-server scenario, to handle multiple clients' requests, it may be necessary to have multiple servers. However, there are times that one server is overloaded by requests from clients for various reasons, and other servers are idle. In this case, it would make sense to redirect some requests to the lightly-loaded servers to achieve better overall performance.

We can use active filter to rectify this situation, by which node level routing decision-making takes place in the presence of some (limited) information about the network level state of congestion. Active filters can determine system topology by exploring the network, then store this information in the nodes on the network. Other active filters use this stored information to derive multi-hop routes across the network.

With the mechanisms described before, e.g., collecting information, communication with peer active filters and abilities of changing network resources, active filters are good

candidates for this kind of task. Since active filters are considered part of an application, they should reroute only the flows that belong to one application.

We can use a simple experiment to illustrate how active filters can balance an application's load by rerouting. The example application has multiple flows that use a virtual network that use a virtual network that has 50% of the bandwidth reserved on each of the three links between the routers. Flows originate from either m8 or m9 and the resource trees on Link 1, 2 and 3 are shown in Figure 5.4. On Link 1, Node 1 corresponds to this application and Node 2 corresponds to some other competing application. Node 3 corresponds to one specific flow of this application, m9 to m2. Node 4 corresponds to another flow of this application, m9 to m4, and is drawn in dotted line, meaning this flow is not known to the scheduler and it will be classified to Node 1. On Link 2, Node 1 and Node 3 are the same as on Link 1, but there are no other applications that have reserved resources. On Link 3, Node 1 again corresponds to our application, and Node 2 represents some other competing application. Node 3 and Node 4 correspond to a flow from m9 to m2 and a flow from m8 to m6 respectively, and they are drawn in dotted lines, meaning that they do not have individual reservations.



(a) (b) (c)

(a) Resource tree on Link1 (b) Resource tree on Link2 (c) Resource tree on Link3

Figure 5.4 Resource trees

The active filter on PC 1 is responsible for one particular flow, m9 to m2, of this application. It knows the bandwidth usage by this flow on Links 1 and 2 by directly monitoring them, and it queries the active filter on PC 3 to get the available bandwidth for

this application on Link 3. Initially, the route flow m9 to m2 passes router PC 1 and PC 2 only (the shortest path). Since the application has a 50% reservation, this flow gets about 50 Mbps throughput. When another flow, m9 to m4, which belongs to this application joins, they share the bandwidth reserved by the application, i.e., each gets about 25 Mbps. As this time, the active filter on PC 1 knows that 50 Mbps are available on Link 2 and, by querying the active filter on PC 3, it understands that the available bandwidth for this application on Link 3 is 50 Mbps. the minimum of these two numbers is larger than what flow m9 to m2 is using, so the active filter on PC 1 makes the decision to reroute flow m9 to m2 through PC 3. Later, when another flow, m8 to m6, which also belongs to the application starts, flow m9 to m2 still goes through PC 3 until flow m9 to m4 finishes, making more bandwidth available on Link 1. At that time, the active filter changes the route for flow m9 to m2 back to its initial route.

5.3 Active Filters in Wireline/Wireless Network

Future wireless media systems will require mobile multimedia communications to support the seamless delivery of voice, video and data with QoS guaranteed. Delivering hard QoS guarantees in the wireless domain is very difficult due to large-scale mobility requirements, limited resources (e.g., relatively low bandwidth) and fluctuating network conditions. In this section, we argue that by using active filters, we can scale flows during periods of QoS fluctuation.

5.3.1 Active Filters

Active filters are active technology, and in our approach they are based on Java coded agents which are capable of being dynamically dispatched to strategic nodes (which should be active nodes, such as base stations, switches, etc.) in the wireline/wireless network, and could automatically scale flows in active nodes during the periods of drastic QoS fluctuation and congestion to seamless deliver audio and video flow to the mobile end users with a smooth change of perceptual quality. Active filters are dispatched, configured and executed at active nodes (here we also consider the mobile end-system as an active node). They are autonomous agents that continuously monitor a flow's available

bandwidth and self-adjust their filtering operations based on the QOS metric via a filter interface to match the available resources at a particular bottleneck node.

5.3.2 Active Node

One of the aims of the thesis is to explore providing applications with a higher degree of programmability to address the QOS control and management in networks. In this section, the programmability means that active node can provide a set of QOS configurable object-level APIs and algorithms for transport, mobility and media scaling. The adaptive and active transport and media scaling algorithm objects are Java program code which are remotely fetched for execution by using a network loader service.

Active filter could exploit the intrinsic scalable properties of multi-layer and multi-resolution audio and video flows and the knowledge of user supplied scaling preferences to actively filter flows at active nodes in the network in order to best utilize the available bandwidth and to seamlessly deliver media with smooth change in the perceptual quality to mobile end users.

Multimedia flows can be represented as multi-layer scalable flows and supported by the semantics of the active network service which can be intelligently and perceptibly scaled-up or scaled-down to match the available resources.

5.3.3 Media Scaling

Media scaling is a technique for the dynamic manipulation of audio and video flows by active filters as flows pass through the active nodes. Media scaling is implemented in active nodes using *filter control objects* (which reside in active node or end system) and active filters (implemented as Java classes). Active filters are Smart Packets written as Java bytecode classes and dispatched to the desired network node (active node) using a filter control algorithms which interacts with an enhanced network loader. Active filters can be dispatched, configured and tuned by filter control (See Figure 5.5).

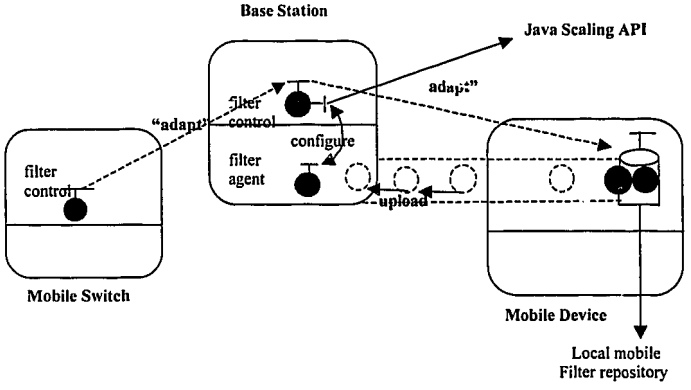


Figure 5.5 Filter Control and Filter Agent Interactions

Active filters are autonomous self-adjusting and could be driven by the *small state* of active node, which means that active nodes use the small state to refresh flow-state (i.e., allocated bandwidth) and filter-state (i.e., instantiated active filters). The media scaling object model is divided into three operational modes:

- *filter control* is a distributed signaling algorithm which comprises of filter control objects. These objects are permanently resident at base stations (active nodes), mobile capable switches (active nodes) and mobile end systems. Filter control objects support a set of methods to *select*, *dispatch* and *configure* active filters;
- *filter instantiation* fetches remote Java bytecode classes and *bootstraps* them into Java VM environment based at active nodes. Once an active filter has been loaded and booted into an active node the local filter control object initiates a *configure* operation to complete the instantiation phase. At this point active filters act autonomously in the flow filtering mode;

- flow filtering algorithms operate in the flow filtering mode where autonomous active filtering algorithms interact with the adaptive service small state mechanism to periodically *tune* flows.

5.3.4 Resource Probing and Automatic Teardown

As part of QOS renegotiation the adaptive network service mechanism resident at the mobile end-systems periodically probes for resources between the end-system and the neighbor active node by sending probe messages (*probe*) toward the active node. These *probe* messages carry the user (or application) desired QOS requirement (such as, bandwidth required for the BL, E1 and E2 layers) for each flow terminated at the user. The active node responds to the *probe* message by issuing an *adapt* message which advertises the explicit rate made available to the user during the next small state refresh interval. This resource management scheme is especially suitable for multicast QOS where individual clients may have different QOS capability. The *adapt* messages are acted as a signal to refresh the small state (flow and filter-state).

Small state disappears if an *adapt* is not received during the refresh interval. This results in de-allocation of network resources and active filters which is called *automatic teardown.*

5.3.5 Media Scaling Operations

Now we go through the operations of the media scaling algorithm. The media scaling process is also demonstrated in Figure 5.6.

(1) Filter Selection. This approach can provide the end user systems with the flexibility to select media scaling algorithms that best suit the application QOS needs and the coding semantics of the transported flow. Such as the following two classes of active filters which are supported for manipulating the rate of MPEG coded video:

- *Selective packet dropping filters* operate on the flow as it traverses the active node to ensure that the appropriate combination of base layer and enhancement layers are forwarded to the proper link to the end users. Selective packet dropping filters only select and drop resolutions, they do not process the media;

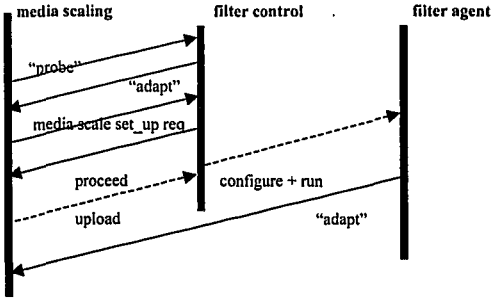


Figure 5.6 Media Scaling Process

- *Dynamic rate shaping filters* are used to adapt the rate of compressed video (MPEG, H261, MJPEG) to the dynamically varying rate constraints of the network environment. Rate shaping filters can shape flows to meet any bandwidth availability but are computationally intensive in comparison to media selectors.

(2) Filter Dispatch. New active filters need to be dispatched under conditions when there is a drastic degradation in the delivered QOS (e.g. network congestion). At this point a filter is selected and dispatched from a filter server resident at the end user system or in the network (such as a *filter service station* if the end system can not offer appropriate active filter). In this case filter control interacts with the filter control object at the designated filtering node and arranges to dispatch the active filter. The transfer of the Java bytecode class is achieved through the interaction of the filter network loader to fetch the program code.

The end system's available resources are indicated by periodic feedback ("*adapt*") messages which defines the specific allocated bandwidth for flows received by the end user. Every *adapt* message provides the end user with state information which could be used to select a new filter and dispatch it into the network. Once a filter has been

dispatched it needs to be bootstrapped in and configured before it is operational and can execute in the active node.

(3) Bootstrap. Once a dispatch acknowledgment message has been received from the target node the bootstrap process is initiated by filter control.

(4) Filter Configuration. After bootstrap is complete filter control configures the active filter by forwarding the filter-spec to the filter. The filter-spec indicates the bandwidth required to support the base and enhancement layers. This allows the remote active filter agent to configure itself and complete the instantiation phase.

(5) Filter Tuning. Active filters are autonomous self-adjusting agents which are driven by the resource level indicated by the *adapt* messages. No interaction with filter control is required for active filters to adjust to changing network conditions. The advertised rate in the adapt messages indicated whether there is sufficient resources to provide enhanced quality to the end user. Two modes of filter tuning are supported. The scaling-down mode informs the active filters to drop enhancement layers. The other mode is the scaling-up mode which is generally invoked when resources free-up.
Resources are allocated to the end user over a route for the duration of the small state refresh time via the *probe/adapt* messages. These message pair periodically probes the communications systems for resources. The adapt messages is used to tune active filters at active node. The available bandwidth advertised in the *adapt* messages is used to adjust the filtering operation of active filters. This has the effect of periodically tuning the filters, e.g., to add or delete a specific enhancement layer, or add or delete coded content or objects.

5.4 Active filters in Real-Time Multicast

Real-time multicast is focusing on delivering streaming data to multiple clients at the same time through network, while adapting streaming data to the variable available bandwidth in multiple paths. Real-time multicast is delivering data through a tree organized network structure. In this approach, it sends data using IP multicast. The real-

time multicast tree is same as the multicast tree that is formed by the multicast routing mechanism. There is explicit ACK packet sent by each client for each data packet it received.

5.4.1 Active node and multicast strategic point

The intermediate node that has the ability to support executing user specified active packet program besides doing traditional routing functions is called *active node*.

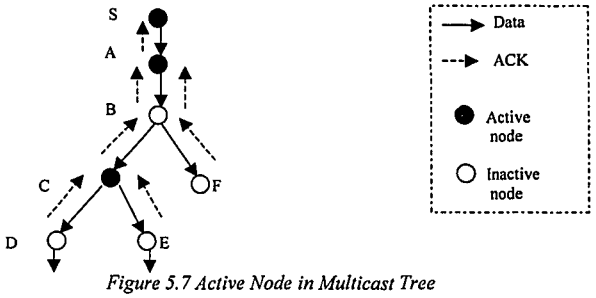


Figure 5.7 Active Node in Multicast Tree

Assume S is one sender of the real-time multicast group. Using S as root to construct a real-time multicast stream distribution tree as in Figure 5.7.

In Figure 5.7, node S, A and C are active node which can support additional function for processing the real-time multicast data stream and the acknowledge packets. These nodes are called *active nodes*. In real-time multicast, we let the sender and all the clients belong to active nodes. Node B and C are where data packets duplicate and forward to different branches in the multicast session. They are called *multicast strategic points*. Node A and node C are active nodes that receive ACKs from more than one downstream node. They make aggregation to ACK packets. **Note** in a real-time multicast tree, the active node is not necessary be the multicast strategic point, vice versa. But it's better for the real-time multicast session situation that all the multicast strategic points are active nodes.

In this approach, we use input/output interfaces to refer to the stream data input/output interfaces in routers. The ACK packets are going through in the reverse direction. And we assume the ACK packets go through the network along the reverse data stream path.

We use downstream and upstream to refer to the relative positions of nodes. For any node X in the real-time multicast tree, if a node Y is on the path between source sender and X, then X is a downstream node of Y, and Y is a upstream node to X. In Figure 5.8, for example, node C, D, E and F are all downstream nodes of node A.

5.4.2 Active Filter Approach

In a multiple-clients streaming application, usually the path properties to the clients are varied in network. Sending stream to multiple clients at same time has the problem of difficulty to adapting one data stream sending rate to multiple clients.

In Figure 5.8, suppose node S wants to send one data stream to C, D and E at the same time. But the paths to node C, D and E have different bandwidth properties. If node S sends data according to the available path bandwidth to node D as 4.5Mbps, it will cause congestion on link between node B and node E. Similar situation happened if node S sends data stream according to the available bandwidth to node C. If node S sends data

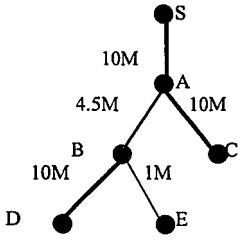


Figure 5.8 Multicast Tree with Different Bandwidth Properties

according to the available bandwidth on the path to node E, which can be supported by all the path, then node C and node E will get lower quality stream than what they can get in the case before.

One solution to this problem is let the stream dynamically changed in the network according to the current downstream capacity. This approach is try to combined active network with the multiple priorities stream encoding technology to making media scaling in the network in order to adapt data stream to different paths.

The sender uses the feedback information from network to adapt the sending speed of the data stream. It tries to let this stream satisfy the "best path", which is the path with largest path available bandwidth. The data stream is sending out through IP multicast. In an active node in the real-time multicast tree, it may drop some packets of the data stream based on the downstream link properties. Because of the possible frequent variation of the available bandwidth in the network, it needs to use feedback information to dynamically detect available bandwidth along each path, and lets the active filter change the stream as the bandwidth variation. Each client sends explicit acknowledgment for each packet it receives. This ACK packet carries additional link information (QoS metrics, such as free buffer size). When these ACK packets pass an active node, the node will make aggregation to these packets and combined its own output links' free buffer size into the produced ACK packet.

In this approach, the requirements to the active network node are:

- Support for injecting active filter program in it.
- Keep small-states for each output links
- Filter streaming data packet based on the small-states in the node

It's not needed that all the network nodes in the real-time multicast tree are active. The more the active node, the more scalability of stream it can supply. It's better to have an active node at a multicast strategic point, so that it can make packet filtering to produce

different streams to different clients when the paths' bandwidths from this point to different clients have great variances.

This approach tries to let the performance degrade gracefully when the percentage of active nodes in the network decreases. When all of the network nodes are non-active, this approach degrades to the End-to-End approach that adapts to one path. This approach is also based on the packet coding and packet filtering mechanism to the data stream. The data stream is coded into packets that contain priority information in them. In the active node, the active filter use this priority information to decide which packet to drop to produce stream "suitable" to each data output link. The result stream level is decided by the small-states associated with each link. Another important aspect of this approach is the *representative aggregation.* For each active node in the real-time multicast tree (include the sender), it usually gets feedback from only one representative from the downstream. And it will use that feedback information to control the data stream to downstream.

5.4.3 Mechanisms

In order to supply feedback to sender and all the active nodes, each client sends explicit acknowledgment for each packet it receives. In this approach, we let each active node act as the representative of all its downstream clients to the neighbor active node in the upstream (In Figure 5.8, node A is the neighbor active node of node C in upstream, node S is the neighbor active node of node A in upstream). For each active node, the ACK packets it received all have the addresses of its neighbor active nodes in downstream (One node can have several neighbor active nodes in downstream). (*Remember that we also call the client side as active node if necessary*).

ACK packet

An ACK packet is generated by a client and sent along the direction in the tree to the sender. The ACK packet used in this approach has the following outstanding fields:

- N*ack* : the acknowledge number

- *Cbuf* : the free buffer size, which indicate at most how many more packets it can receive now.

When the ACK packet is passed through an active node, the active node will try to aggregate this packet with other same acknowledge number ACK packets. The active node will also compare the local node information with the information in the packet and make changes to the free buffer size field if needed. Finally it will put its own address into the output ACK packet produced by this aggregation.

Assumption to Filters

At any time when the upstream data stream increase, the filter can still keep the "same level" output stream by dropping more data.

In Figure 5.9, S*i* is the data stream come from upstream, S*o* is the data stream output through the filter to downstream. S*drop* is the part of data stream dropped by the filter. So, S*i* = S*o* + S*drop*. When S*i* is increased, the packet filter can still guarantee the S*o* get same data stream as before. (Although S*o* may have larger packet sequence number gap). This requires that when the source increases the data stream level, it will only add packets into the original data stream. This assumption guarantees that one links bandwidth increase wouldn't affects other links data stream.

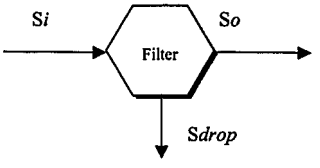


Figure 5.9 Filter Assumption (Si = So + Sdrop)

Congestion Detection Module (CDM)

From the server to downstream clients, the data stream will be filtered at each active node to stream according to the feedback get back from the corresponding downstream path. In this section we will describe the estimation part of work at active node to supply

information to packet filter. There will be a *Congestion Detection Module* for each output link in an active network node.

In Figure 5.10, we show the internal data flow in an active node with only two output interfaces in one real-time multicast session. The input data stream to the active node go through the filter and produce two streams (In Figure 5.10, the filter's output to interface 2 is same to the input data stream, and output to interface 1 is half of the input data stream). The output stream of the Filter will be the input of CDM. Then the packet of the output of CDM will go to the output interface buffer. The ACK packets are first forwarded to the CDM in the same interface. Then the CDM will forward ACK packets to the Aggregation Module in the active node, which will aggregate ACKs packets come from all the output interfaces in the real-time multicast session. The output of the Aggregation Module will be the ACK packets sent to upstream.

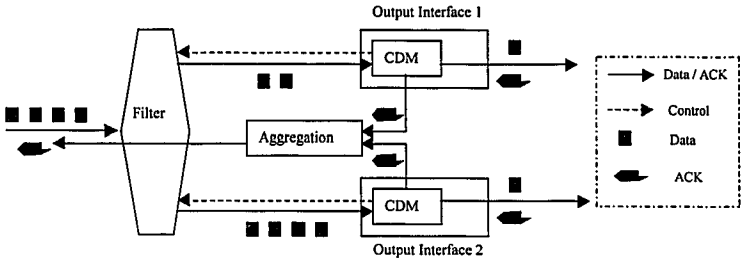


Figure 5.10 Internal Data Flow in an Active Node

In order to let the packet filter produce the proper output to the downstream, the Congestion Detection Module needs to give the packet filter enough information to let it make decision on how much input stream will be the output stream to this output interface.

Acknowledge aggregation

In order to give correct feedback information to upstream, each active node needs not only to make packet filtering but also to make ACKs aggregation. Because of the assumption we made before, what each active node want to let the upstream node know is the best stream it can support currently. So it choose the "best" path's ACK packets to forward to upstream.

The address of ACK packets to upstream will use address of the active node. This is used in order to guarantee these ACKs are looked by upstream node as come from one representative.

Multiple ACKs in one output link

When a multicast strategic point is not an active node, there will be several ACKs go to upstream through one link. So it's possible for an interface of an active node in the upstream gets more than one ACK packet streams.

The interface in the active node will be able to distinguish them by looking their source address. Before making estimation in CDM, the interface can make aggregation for them before. This aggregation can also follow the policy that choose the "best" one, or use a simple aggregation mechanism as forwarding the first arrived one for each sequence number.

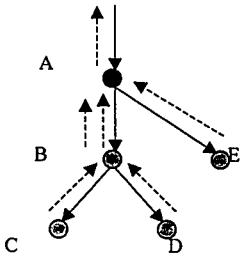


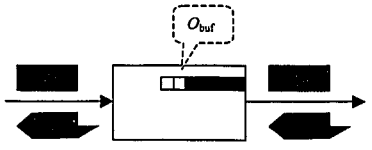
Figure 5.11 ACKs' Aggregation

In Figure 5.11, one of the interface of active node A will receive ACKs from both C and D. These ACKs will be aggregate first in this interface, then it will be aggregated again in node A with ACKs from node E.

Single input/output active node

For the active node with only one input link and one output link in the current real-time multicast tree, there will be no ACKs aggregation work. But in order to use the free buffer size information, this active node will need to compare its free buffer size with the value carried by the ACK packets come from downstream.

In Figure 5.12, the free buffer size value (C_{buf2}) in the ACK packet to upstream will be smaller one between the originally value in the ACK packet come from downstream (C_{buf1}) and free buffer size in this output interface (O_{buf})



ACK : $N_{ack} + C_{buf1}$
ACK': $N_{ack} + C_{buf2}$
O_{buf} : Free buffer size of output link 1
$C_{buf2} = \text{Min}\ (C_{buf1}, O_{buf})$

Figure 5.12 Single Input/Output Active Node

Chapter 6
Simulation by Using Grasshopper

Grasshopper is a mobile agent platform that is built on top of a distributed processing environment [IKV98]. In this way, an integration of the traditional client/server paradigm and mobile agent technology can be achieved. Grasshopper is developed compliant to the first mobile agent standard of the *Object Management Group* (OMG), i.e. the *Mobile Agent System Interoperability Facility* (MASIF). The MASIF standard has been initiated in order to achieve interoperability between mobile platforms of different mobile agent platforms of different manufacturers. In this Chapter, we will introduce Grasshopper, and how we use Grasshopper to simulate load-sensitive rerouting. Section 6.1 devotes to the introduction of distributed agent environment; Section 6.2 describes the communication concepts in Grasshopper; and the simulation of load-sensitive rerouting by using Grasshopper is presented in Section 6.3.

6.1 Distributed Agent Environment

This section describes the structure of the Grasshopper *Distributed Agent Environment* (DAE). The DAE is composed of regions, places, agencies and different types of agents. Figure 6.1 depicts an abstract view of these entities.

6.1.1 Agents

Two types of agents act in the Grasshopper context, i.e., *stationary agents* and *mobile agents*. As we already introduced in Chapter 3, mobile agents are able to move from one physical network location to another, they can migrate to the desired communication peer and take advantage of local interactions. In contrast to mobile agents, stationary agents do not have the ability to migrate actively between different network locations. Instead, they are associated with one specific location.

6.1.2 Agencies

An Agency is the actual runtime environment for mobile and stationary agents. At least one agency must run on each host that shall be able to support the execution of agents. A Grasshopper agency consists of two parts, i.e., the core agency and one or more places.

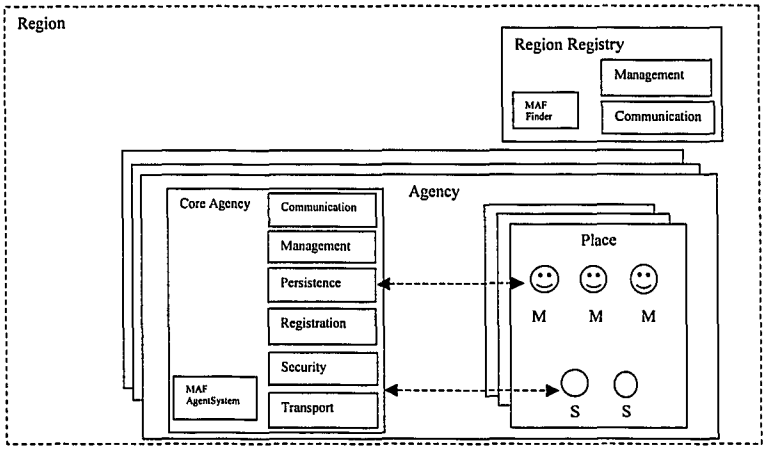


Figure 6.1 Hierarchical Component Structure

Core Agency

Core Agencies represent the minimal functionality required by an agency in order to support the execution of agents. The following services are provided by a Grasshopper core agency :

- **Communication Service :** This service is responsible for all remote interactions that take place between the distributed components of Grasshopper, such as location-transparent inter-agent communication, agent transport, and the localisation of agents by means of the region registry. All interactions can be performed via CORBA IIOP, Java RMI, or plain socket connections.
- **Registration Service :** Each agency must be able to know about all currently hosted agents and places, on the one hand for external management purposes and on the

other hand in order to deliver information about registered entities to hosted agents. The registration service is developed to achieve this.

- **Management Service :** Management services are developed to allow the monitoring and control of agents and places of an agency by external (human) users.
- **Transport Service :** This service supports the migration of agents from one agency to another. At the destination agency, the agent continues its task processing exactly at the point where it has been interrupted before the migration.
- **Security Service :** Grasshopper supports two kinds of security mechanisms, i.e. *external security* and *internal security*. External security protects remote interactions between the distributed Grasshopper components, i.e., agencies and region registries. Internal security protects agency resources from unauthorised access by agents. This is achieved by authenticating and authorising the user on whose behalf an agent is executed.
- **Persistence Service :** The Grasshopper persistence service enables the storage of agents and places on a persistent medium. In this way it is possible to recover agents or places when needed, e.g., when an agency is restarted after a system crash.

Places

A place provides a logical grouping of functionality inside an agency. The name of the place should reflect its purpose. For example, in every egancy exists by default a place named `InformationDesk`. Every agent with no determined place is transported to the `InformationDesk` where it can look for further information.

6.1.3 Regions

The region concept facilitates the management of the distribued components in the Grasshopper environment, i.e., agencies, places, and agents. Agencies as well as their places can be associated with a specific region, i.e., they are registered within one region registry. Each registry automatically registers each agent that is currently hosted by an agency associated with the region. If an agent moves to another location, the corresponding registry information is automatically updated. A region may comprise all agencies belonging to a specific company or organisation, thus facilitating its management.

6.2 Communication Concepts

The section explains the communication concepts of the Grasshopper platform. These concepts are realised by means of the Grasshopper *communication service (CS)* which is an essential part of each core agency. The communication service allows location-transparent interactions between agents, agencies, and non-agent-based entities.

6.2.1 Multi-protocol Support

Remote interactions are generally achieved by means of a specific protocol. The CS supports communication via *Internet Inter-ORB Protocol* (IIOP), *Java's Remote Method Invocation* (RMI), and *plain socket connections*. To achieve a secure communication, RMI and the plain socket connection can optionally be protected with the *Secure Socket Layer* (SSL).

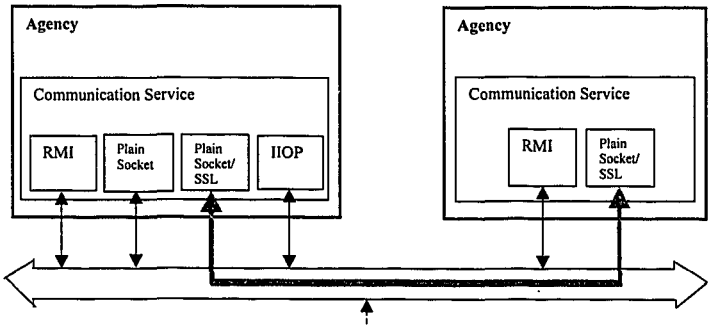


Figure 6.2 Multi-Protocol Support

6.2.2 Location Transparency

On the one hand the communication service is used by the Grasshopper system, e.g., for agent transport or for locating entities within the DAE. On the other hand, agents can use the CS to invoke methods on other agents. This is done location-transparently, i.e., the agent need not care about the location of the desired communication peer. Within the agent code, remote method invocations look exactly like local method invocations on objects residing on the same Java Virtual Machine.

6.3 Simulation by Using Grasshopper

In this section, we will use Grasshopper mobile agent platform to simulate load-sensitive rerouting which we discussed in Chapter 5, Section 5.2.

We created three agencies to simulate three active routers. And we also create one mobile agent in Agency 1. The Grasshopper testbed for this simulation is shown in Figure 6.3. We use this mobile agent in Agency 1 to simulate an active filter. By using this mobile agent, node level routing decision-making can take place in the presence of some (limited) information about the network level state of congestion because active filters can determine system topology by exploring the network. We use this mobile agent in Agency 1 to collect the bandwidth information of Agency 2 and Agency 3, compare these two bandwidth information, then make the routing decision.

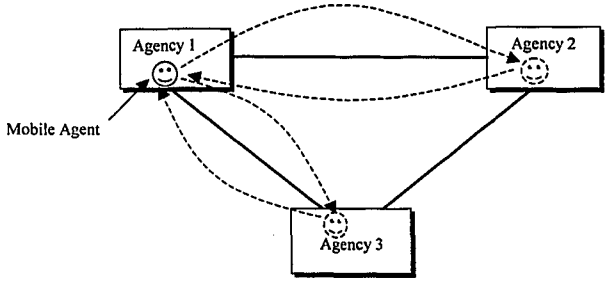


Figure 6.3 Grasshopper Testbed

We implemented a simple experiment to illustrate how this mobile agent can balance an application's load by rerouting. We create a mobile agent called BandwidthCheckAgent, this agent moves to a remote agency (Agency_2) specified by the application. At the remote agency, it collects information about the free bandwidth (here, we let the mobile agent to check the free memory) and returns. Back home, it pops up a window showing the free bandwidth of the remote agency. Then the agent moves to another remote agency (Agency_3) and collects information about the free bandwidth (here, also means checking

free memory) and returns. It can compare the two bandwidth information, then makes the routing decision, i.e., it will route the packet to the router which has the larger bandwidth information.

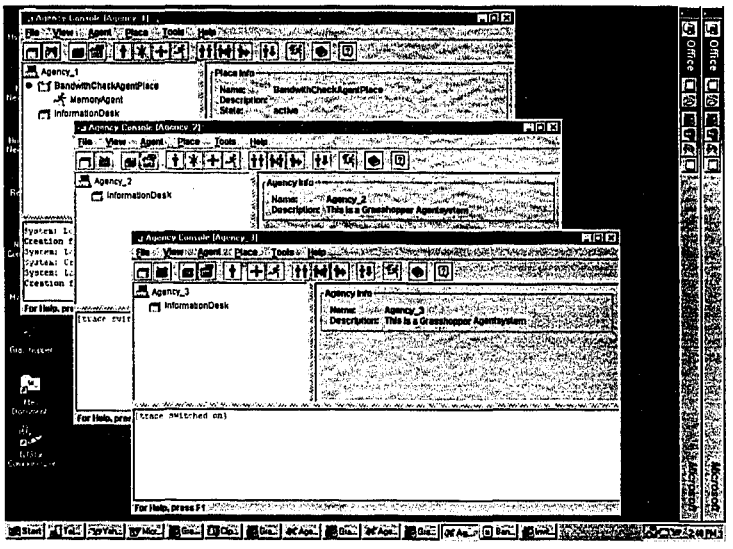


Figure 6.4 Agency 1, Agency 2 and Agency 3

Figure 6.5 GUI of BandwidthCheckAgent

Chapter 7
Evaluations and Conclusions

This chapter gives the evaluation and conclusions of our approach. Section 7.1 gives the evaluation of active filters in real-time resource management and load-sensitive flow rerouting. Section 7.2 evaluates the active filters in wireline/wireless network. Evaluation of active filters in real-time multicast is discussed in Section 7.3.

7.1 Evaluation of active filters in real-time resource management and load-sensitive flow rerouting

In this approach, we introduced the concept of *active filter*, a code segment that applications or service providers inject into the network to assist in the runtime management of the network resources that are allocated to them. Our active filter architecture was driven by two requirements. First, users should be able to tailor resource management so they can optimize their notion of quality of service. Second, since active filters execute inside the network, they can quickly respond to changes in the network conditions. We described the programming interface that active filters can use to monitor the network conditions, e.g. queue status and bandwidth of the flows they are responsible for, and to modify resource use, e.g. changing reservations, selective packet dropping or rerouting.

We described a number of active filters addressing problems such as congestion control for video streaming, tracking down non-conformant traffic sources, and balancing of traffic load. While some active filters operate in a purely local fashion, others require coordinated actions by active filters running on multiple routers. While none of the examples provides necessarily the best, or even a complete, solution to these problems, they do illustrate that our programming interface is rich enough to support a broad range of resource management actions. Further research will compare the benefits of being able to make customized resource management decisions inside the network, with the

additional complexity active filters introduce.

7.2 Evaluation of active filters in wireline/wireless network

In this approach, *active filters* are active technology which are based on Java coded agents, are capable of being dynamically dispatched to strategic nodes (which should be active nodes, such as base stations, switches, etc.) in the wireline/wireless network, and could automatically scale flows in active nodes during the periods of drastic QoS fluctuation and congestion to seamless deliver audio and video flow to the mobile end users with a smooth change of perceptual quality. Active filters are dispatched, configured and executed at active nodes (here we also consider the mobile end-system as an active node). They are autonomous agents that continuously monitor a flow's available bandwidth and self-adjust their filtering operations based on the QOS metric via a filter interface to match the available resources at a particular bottleneck node.

One of the key performance issues related to this technology is the time taken to dispatch, bootstrap and configure new agent over the wireless/wireline interfaces. Another important performance concern relates to the performance penalty paid by flows as they are filtered at switches and base stations. The amount of delay introduced by such operations as flows traverse active filters is dependent on the computational complexity of the filter.

We have coded a selective packet dropping filter in Java that drops either E1 (i.e., P pictures) and E2 (i.e., B pictures) frames based on the available resources (see Appendix). Selective packet dropping filters do not process the media, therefore, the algorithms have the least impact of all the proposed filters.

Selective packet dropping filters are computationally simple and an attractive type of filter, which can significantly reduce the data rate of a video stream without degrading its quality to an unacceptable levels. Some of the positive attributes of selective packet dropping filters is the small processing delays incurred at the base stations during media scaling. Only headers of incoming frames are examined and the frame dispatched during

media scaling. Instantiation time is also modest dues to the length of media selector's bytecode which is modest in comparison with other filters like dynamic rate shaping filters.

A major disadvantage of the selective packet dropping filters is that is lacks the ability to set an incoming bit rate to a desired level; it does not operate over a continuum of available bandwidth but at discrete bandwidth intervals. In this scheme selective packet dropping filters operate at three distinct discrete rates. There is a major trade-off between a filter complexity and continuous scale of bandwidth.

7.3 Evaluation of active filters in real-time multicast

As we introduced in Section 5.4, real-time multicast is focusing on delivering streaming data to multiple clients at the same time through network, while adapting streaming data to the variable available bandwidth in multiple paths. It is delivering data through a tree organized network structure. In our approach, it sends data using IP multicast.

In our approach, we only assume one QoS dimension (bandwidth), one thing we want to show in the future is policy on multiple QoS dimensions, which may need to cause more complex policy description to make feedback information aggregation in network node. Also, if we consider multiple QoS dimensions, there will exist problem on that if we should encode multiple QoS information into the packet, and if we should add more particular requirements to the active node's work.

We did not consider to keep the real-time multicast result stable, which means users may want to get a more stable stream instead of a frequently changed stream, this problem remains as a further research consideration. For example, the end-system users may want to see either mono-color or color, they may not want to switch frequently.

BIBLIOGRAPHY

[ABG+97] D. Scott Alexander, Bob Braden, Carl A. Gunter, Alden W. Jackson, Angelos D. Keromyties, Gary J. Minden, David Wetherall. "Active Network Encapsulation Protocol (ANEP)". RFC Draft. July 1997.

[ATW+90] D. Anderson, S. Tzou, R. Whabe, R. Govindan and M. Andrews, "Support for Continuous Media in the DASH System", In Proceedings of the 10th ICDCS, Paris, France, May 1990.

[BCZ97] Samrat Bhattacharjee, Kenneth L. Calvert, Ellen W. Zegura. "An Architecture for Active Networking". Georgia Institute of Technology. High Performance Networking (HPN'97), White Plains, NY, April 1997.

[BHN+97] Joachim Baumann, Fritz Hohl, Nikolaos Radouniklis, Kurt Rothermel, and Markus Strer. "Communication concepts for mobile agent systems." In ma97, Berlin, Germany, April 1997.

[BKR98] Jonathan Bredin, David Kotz, and Daniela Rus. "Market-based resource control for mobile agents." In Proceedings of Autonomous Agents. May 1998.
ftp://ftp.cs.dartmouth.edu/pub/kotz/papers/bredin:market.ps.Z.

[BWP98] Bieszczad, A., White, T., Pagurek, B. "Mobile Agents for Network Management." In IEEE Communications Surveys, September, 1998.

[CBZ+98] Kenneth L. Calvert, Samrat Bhattacharjee, Ellen Zegura and James Sterbenz. "Directions in Active Networks." Technical Report GIT-CC-98-16, College of Computing, Georgia Institute of Technology, 1998.

[CFK+98] Prashant Chandra, Allan Fisher, Corey Kosak, T. S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, Hui Zhang. "Darwin: Resource Management for Value-Added Customizable Network Service", extended draft of the ICNP paper, 1998

[CHK97] David Chess, Colin Harrison, and Aaron Kershenbaum. "Mobile agents: Are they a good idea?" In mob-obj-sys, pages 46-48. Springer-Verlag, April 1997. Incs1222.

[CKV+99] Campbell, A.T., Kounavis, M.E., Vicente, J., Villela, Miki, K. and H. De Meer, "A Survey of Programmable Networks", ACM SIGCOMM Computer Communication Review, April 1999.

[DARPA96] DARPA "Active Network Program"
http://www.darpa.mil/ito/research/anets/projects.html, 1996

[FJ93] Sally Floyd and Van Jacobson. "Random early detection gateways for congestion avoidance." IEEE/ACM Transactions on Networking, 1(4): 397-413, August 1993.

[GY95] German Goldszmidt, Yechiam Yemini. "Distributed Management by Delegation". In Proceedings of the 15th International Conference on Distributed Computing Systems, 1995.

[Hehmann91] D. Hehmann et.al. "Implementing HeiTs: Architecture and Implementation Strategy of the Heifelberg High-Speed Transport System", Proceedings of 2nd International Workshop on Network and Operating Systems Support for Digital Audio and Video, Heidelberg, Germany, November 1991.

[HMA+98] Michael Hicks, Jonathan T. Moore, D. Scott Alexander, Carl A. Gunter, and Scott M. Nettles. "PLANet: An Active Internetwork". Department of Computer and Information Science. University of Pennsylvania. August 2, 1998.

[HSF93] D. Hoffman, M. Speer and G. Fernando. "Network Support for Dynamically Scaled Multimedia Data Streams", Proceedings of 4th International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster University, Lancaster, UK. 1993, pp.251-262

[HSF93] D. Hoffman, M. Speer and G. Fernando. "Network Support for Dynamically Scaled Multimedia Data Streams", Proceedings of 4th International Workshop on Network and Operating System Support for Digital Audio and Video. Lancaster University, Lancaster, UK, 1993, pp.251-262.

[IKV98] IKV++/GMD. Architecture of grasshopper. http://www.ikv.de/products/grasshopper/content.html, 1998

[KMH+98] A. B. Kulkarni, G. J. Minden, R. Hill, Y. Wijata, A. Gopinath, S. Sheth, F. Wahhab, H. Pindi and A. Nagarajan. "Implementation of a Prototype Active Network". Department of Electrical Engineering and Computer Science, University of Kansas. 1998.

[LM97] Dong Lin and Robert Morris. "Dynamics of Random Early Detection." In Proceedings of the SIGCOMM '97 Symposium on Communications Architectures and Protocols, pages 127-137, Cannes, August 1997. ACM

[LWG98] Ulana Legedza, David J. Wetherall, and John Guttag. "Improving the performance of distributed applicaions using active networks." IEEE

Infocom '98, 1998.

[PPA+92] J. Pasquale, G. Polyzos, E. Anderson and V. Kompella. "The Multimedia Multicast Channel", Proceedings of 3rd International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, California, 1992, pp.185-196

[PPA+93] J. Pasquale, G. Polyzos, E. Anderson and V. Kompella. "Filter Propagation in Dissemination Trees: Trading off Bandwidth and Processing in Continuous Media Networks", Proceedings of 4th International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, California, 1993, pp.269-278.

[PPV98] G. Parulkar, C. Papadopoulos and G. Varghese. "An error control scheme for large-scale multicast applications". Infocom '98, 1998.

[RRV93] S. Ramanathan, P. Venkat Rangan and H. Vin, "Frame-Induced Packet Discarding: An Efficient Strategy for Video Networking", Proceedings of 4th International Workshop on Network and Operating System Support for Digital Audio and Video. Lancaster University, Lancaster, UK, 1993, pp.175-186

[SC] Akhil Sahai, Christine Morin. "Towards Distributed and Dynamic Network Management". INRIA- IRISA. Campus de Beaulieu. 35042, Rennes CEDEX, France.

[SCF+94] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. "RTP: A Transport Protocol for Real-Time Applications", Internet-Draft, draft-ietf-avt-rtp-05.txt, July 1994.

[SFC+] J.M. Smith, D. J. Farber, C.A. Cunter, S. M. Nettles, Mark E. Segal, W. D. Sincoskie, D. C. Feldmeier and D. Scott Alexander. "SwitchWare: Towards a 21st Century Network Infrastructure". CIS Department, University of Pennsylvania.

[SZJ+] Beverly Schwartz, Wenyi Zhou, Alden W. Jackson, W. Timothy Strayer, Dennis Rockwell, Craig Partridge. "Smart Packets for Active Netwworks". BBN Technologies.

[Tran97] N. Tran. Mobile Agent Assisted Adaptation in Video on Demand. Master Thesis 1997. University of Illinois at Urbana-Champaign

[TW96] David Tennenhouse and David Wetherall. "Towards active network architecture." In Computer Communication Review, 26(2): 5-18, April 1996.

[Wireless] Wireless ATM Voice/Data Project. URL http://www.ittc.ukans.edu/Projects/Wireless_ATM

[WGT96] David J. Wetherall, John V. Guttag and David L. Tennenhouse. "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols". Software Devices and Systems Group. Laboratory for Computer Science, Massachusetts Institute of Technology.

[Yeadon96] Quality of Service for Multimedia Communications. Ph.D. Thesis, Lancaster University. May 1996

[YGH+96] Nicholas Yeadon, Francisco Garcia, David Hutchison and Doug Shepherd. "Filters: QoS Support Mechanisms for Multipeer Communications". In IEEE Journal on Selected Areas in Communications (JSAC) forthcoming issue on Distributed Multimedia Systems and Technology, 3rd Quarter, 1996.

[YS96] Yechiam Yemini and Sushil da Silva. "Towards Programmable Networks". Department of Computer Science, Columbia University. April 1996.

[ZD94] Bruce Zenel, Dan Duchamp. "Intelligent Communication Filtering for Limited Bandwidth Environments". Computer Science Department, Columbia University. 1994

[ZDE+93] L. Zhang, S. Deering, D. Estin, S, Shenker and D. Zappala, "RSVP: A New Resource Reservation Protocol", IEEE Network, vol.7, pp8-18, Sept. 1993.

1 Agent.java

```
// File: Agent.java

import java.lang.*;
import java.util.*;

public abstract class Agent extends Thread {
 Vector vargs;

 public void setArguments(Vector ar) {
  vargs = ar;
 }
 native void testnative();

 public void run() {
  //Here is where the args supplied by setArguments are used
  //For example, the filter agent might say args[1] = <upstream host>
  //and so on..
  try {
   Thread.sleep(10000);
  } catch(InterruptedException e) {
   System.out.println("Exception " + e);
  }
  System.out.println("Agent Started");
 }
}
```

2 FilterController.java

```
// File: FilterController.java

import java.lang.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class FilterController extends Thread {
static final int CONFIRM = 7777;
static final int sigPort = 5510;

//  --- Instance variables ---

UDPSockManager mgr = null;
MediaSelector  mySelector;
int filterSetPort;

//  ---- Methods ----

public FilterController(MediaSelector ms, int fsp) {

      mySelector    = ms;
      filterSetPort  = fsp;
}
```

```
public void run() {
  String bs = " ";   // to be overwritten
  mgr = new UDPSockManager();
  StringPackager packager = new StringPackager(bs);
  while(true) {

     // receive connection request from a base station...
     System.out.println("Filter Controler>Listening on port : "+ filterSetPort);
     int filterType = mgr.recv(filterSetPort, packager);
     System.out.println("Filter Controler>Request received to setMS to "+ filterType);
     bs = packager.getString();

     // Set Filter Type;
     mySelector.setNewType(filterType);
     mySelector.raiseNewFilter();
     System.out.println("Filter Controller>Filter Type was set to "+ filterType
                +" for Agent #"+ (filterSetPort-5600)+ " on host "+bs);
          // send confirmation message...
          mgr.send(bs, sigPort, CONFIRM);
 }
}
}
```

3 FilterDaemon.java

```
// File: FilterDaemon.java

import java.net.*;
import java.io.*;
import java.lang.*;
import java.util.*;

public class FilterDaemon  {
//    Constant Declarations

static final int sigPort = 5600;  // for receiving agent instantiation
                          // requests from the BS
static final int fsPort  = 5115;  // for signalling with the
                          // filter server

//    ---- the main() method ----
public static void main (String args[]) throws SocketException  {
   if (args.length != 1) {
     System.out.println("Usage: java FilterDaemon <filter server>");
     return;
   }

   UDPSockManager mgr = new UDPSockManager();
   StringPackager p   = new StringPackager();
   StringPackager q   = new StringPackager();
   int numOfAgents = 0;
   while(true) {
     int getFrom = mgr.recvIPintf(sigPort, p, q);
```

```
    int UDPport   = (int)(getFrom/10);
    int filterType = (int) getFrom%10;
    String dpathHost = p.getString();
    String mobileHost = q.getString();
    numOfAgents++ ;              // increase the number of agents...
    String[] av = new String[5];    // av:  array of arguments passed to the
                                //      media scaling filter

    av[0] = Integer.toString(UDPport);
    av[1] = Integer.toString(filterType);
    av[2] = dpathHost;
    av[3] = Integer.toString(sigPort+numOfAgents);
    av[4] = new String(mobileHost);

    System.out.println("Filter daemon> Paramaters passed to Agent #" + (numOfAgents-1)+ " : "
                + av[0] +" "+ av[1]+" "+av[2]+" "+av[3]+" "+av[4] );
    Vector arguments = new Vector(5);        // convert array to vector
          MakeVector myVector = new MakeVector();
          arguments = myVector.parse(av);
    System.out.println("Filter daemon> Passing vector of length " + arguments.capacity());

    // Loading the filter agent...
          NetworkLoader myLoader = new NetworkLoader( args[0], fsPort);
    Class myClass = myLoader.loadClass("MediaSelector", true);
    if (myClass == null) {
              System.out.println("Filter Daemon> No Class Loaded");
          }
          else {
              try {
                  Object myObject = myClass.newInstance();
                  if (myObject instanceof Agent) {
                 Agent myAgent = (Agent)myObject;
                 myAgent.setArguments(arguments);
                 myAgent.start();
                 System.out.println("Filter Daemon> Successfully started agent");
            }
               } catch(Exception e) {
                 System.out.println("Filter Daemon>Failed to create Object from Class");
                 System.exit(1);
                }
          }
}
}
}
```

4 FilterServer.java

```
// File: FilterServer.java

import java.lang.*;
import java.net.*;
import java.io.*;

public class FilterServer {
```

```
private static final int sending = 0;
private static final int sent = 1;
private static int state = sending;
public static void main (String args[]) {
 //Default port number
 int Port = 1115;

 //**Initialize Server side Socket. If port is in use try to bind to another
 //local port.
 while(true) {
 ServerSocket servsock = null;
 try {
  servsock = new ServerSocket(Port);
  System.out.println("Server: listening on port " + Port);
 } catch (IOException e) {
  System.out.println("> Could not listen on port: " + Port+ ", " + e);
  System.out.println("> Trying for another port");
  Port++;
  System.out.println("> Server Sig Port: " + Port);
 }

 //**Initialize Client Accept Socket
 Socket dclient = null;
 try {
  dclient = servsock.accept();
 } catch (IOException e) {
  System.out.println("> Accept failed on " + Port + ", " + e);
  System.exit(1);
 }

 //**Define Input and Output Streams on the Client Accept Socket
 try {
  BufferedReader is = new BufferedReader(new InputStreamReader(dclient.getInputStream()));

 // PrintStream os = new PrintStream(new BufferedOutputStream(dclient.getOutputStream()));
  String inline, outline;
  byte[] bret = null;
  String Classfile = null;
 while ((inline = is.readLine()) != null) {

// inline = is.read();
        Classfile = inline.concat(".class");
        System.out.println("Server: File Requested " + Classfile);

        //**<Sending Begins....>
        //**Write the File Input Stream and File Output Streams
        FileInputStream fis = new FileInputStream(Classfile);
        int filesize = fis.available();
        bret = new byte[filesize];
        while (fis.read(bret) != -1) {
          if (bret != null) {
            System.out.println("Server: Bytecode Read");
            //os.write(bret, 0, filesize);
            //os.flush();
            state = sent;
            System.out.println("Server: Agent Sent");
```

```
          }
        }
    }

// os.close();
    is.close();
    dclient.close();
    servsock.close();
   } catch (FileNotFoundException e) {
    System.out.println("> Could not find requested file");
   } catch (IOException e) {
    System.out.println("> IOException: " + e);
    e.printStackTrace();
   }
 }
}
}
```

5 SelectivePacketDropper.java

```
// File: SelectivePacketDropper.java

import java.lang.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class SelectivePacketDropper extends Agent {

//   Constant Declarations
static final int NEW_CONNECTION = 100;
static final int HAND_OVER = 101;
static final int BUFFER_SIZE = 65535;        // maximum AAL5 - UDP packet size
static final int sigPort = 5500;

//    SelectivePacketDropper
static final int DROPB    = 15;
static final int DROPP    = 16;
static final int DROPI    = 17;
static final int SCENE    = 18;
static final int MOTION   = 19;
static final int NOFILTER = 20;

//    Header identifiers
static final int seqHeader    = 1;
static final int gopHeader    = 2;
static final int picHeader    = 3;
static final int eomHeader    = 4;
static final int picHeader_I  = 5;
static final int picHeader_P  = 6;
static final int picHeader_B  = 7;
//      ---- Instance Variables ----
Vector arguments = new Vector(5);
byte[] inputBuffer = null;
```

```
UDPSockManager mgr = new UDPSockManager();
int videoInPort  ;                  // The ports that video is received
int videoOutPort ;                  // from/sent to

// Lancaster Header Elements...
int Header;
int Pad;
int SeqNum;
int TimeStamp;
int SegIndex;

// Filter Controller Thread...
public FilterController  filterctl;
public int filterType = 0;
public int newFilterType = 0;
public boolean newFilterFlag = false;

//      ---- Methods ----
public void setArguments(Vector aVector) {
    arguments = aVector;
}
public void setType(int x) {
    filterType = x;
}
public void setNewType(int x) {
    newFilterType = x;
}
public void raiseNewFilter() {
    newFilterFlag = true;
}
public void dropNewFilter() {
    newFilterFlag = false;
}
public void run()  {
   System.out.println(" Starting Media Selector ...");
   DatagramSocket videoInSoc  = null;  // Video input and output datagram
   DatagramSocket videoOutSoc = null;  // Socket

   // Create the datagram sockets ...
   try {
       videoInSoc  = new DatagramSocket();
   } catch (java.net.SocketException e) {
     System.out.println("> Could not create socket: Video In, " + e);
     System.exit(1);
   }
   try {
       videoOutSoc  = new DatagramSocket();
   } catch (java.net.SocketException e) {
     System.out.println("> Could not create socket: Video Out, " + e);
     System.exit(1);
   }

   // Parse the arguments ...
   int  videoOutPort  = Integer.parseInt(String.valueOf(arguments.elementAt(0)));
        filterType    = Integer.parseInt(String.valueOf(arguments.elementAt(1)));
```

```
// filterType = 0;
String dpathHost   = String.valueOf(arguments.elementAt(2));
int  filterSetPort  = Integer.parseInt(String.valueOf(arguments.elementAt(3)));
System.out.println("Filter Agent> Arguments received : "+videoOutPort+ " " + filterType +" "
            +dpathHost+ " " + filterSetPort);

// Start the filter Type Controlling Thread...
filterctl = new FilterController(this, filterSetPort);
filterctl.start();

// determine the video Input port
videoInPort = videoInSoc.getLocalPort();
System.out.println("Filter Agent> Video Input Port: " + videoInPort);

// and send it to the BS
mgr.send(dpathHost, sigPort, videoInPort);
System.out.println("Filter Agent> response sent to host: "+ dpathHost);

if (videoOutPort != 0)
   System.out.println("Filter Agent> Video Output Port " + videoOutPort);
for ( int count = 0; count >= 0; count++) {       // infinite loop
  DatagramPacket packet = null;
  if (videoInSoc != null && videoOutSoc != null) {
  try {

        // receive and process the packet ...
        inputBuffer = new byte[BUFFER_SIZE];
              packet = new DatagramPacket(inputBuffer, BUFFER_SIZE);
        int stamp1 = (int) System.currentTimeMillis();
        videoInSoc.receive(packet);
        int length = packet.getLength();
        if (count == 0)
          System.out.println("Filter Agent> Data are being received from Datapath Host ");

        //  Intermediate additional filtering
        ByteArrayInputStream Barray = new ByteArrayInputStream(inputBuffer);
              DataInputStream dbs = new DataInputStream(Barray);

        // read the Lancaster Header fields
        Header      = dbs.readUnsignedShort();
        Pad         = dbs.readUnsignedShort();
        SeqNum      = dbs.readInt();
        TimeStamp   = dbs.readInt();
        SegIndex    = dbs.readInt();

        //  change the filter type if needed...
        if ( (newFilterFlag == true) &&
           (Header == seqHeader)  &&
           (SegIndex == 1) )  {
             filterType = newFilterType;
             dropNewFilter();
           }
        // And set the Media Scaling type in the Lancaster Header
        Pad = filterType;
        ByteArrayOutputStream b_os = new ByteArrayOutputStream();
        DataOutputStream d_os     = new DataOutputStream(b_os);
```

```
d_os.writeShort(Header);
d_os.flush();
d_os.writeShort(Pad);
d_os.flush();

        byte[] tempBuf = b_os.toByteArray();
        ByteArrayInputStream b_is = new ByteArrayInputStream(tempBuf);
        b_is.read(inputBuffer, 0, 4);

        // Set the datapath host address
        InetAddress addr = InetAddress.getByName(String.valueOf(arguments.elementAt(4)));

int filter_action = 0;
switch(filterType) {
   case 0: filter_action = NOFILTER; break;
   case 1: filter_action = DROPB   ; break;
   case 2: filter_action = DROPP   ; break;  // meaning Ps and Bs
   default : System.out.println("Filter Agent>Invalid type of filter"); break;
}

        //Now a switch has to be done on the header...
        switch (Header) {
           case(seqHeader):
           case(gopHeader):
           case(picHeader):
           case(eomHeader):
           case(picHeader_I):
            if (filter_action == DROPI) {
               break;
            } else {
        packet = new DatagramPacket(inputBuffer, length, addr, videoOutPort);
        videoOutSoc.send(packet);

        // System.out.println("transmitting I frame/SH/GH");
            }
            break;
           case(picHeader_P):
            if (filter_action == DROPP) {
               break;
            } else {
               packet = new DatagramPacket(inputBuffer, length, addr, videoOutPort);
            videoOutSoc.send(packet);
            // System.out.println("transmitting P frame");
            }
            break;
           case(picHeader_B):
            if (filter_action == DROPB || filter_action == DROPP) {
               break;
            } else {
               packet = new DatagramPacket(inputBuffer, length, addr, videoOutPort);
            videoOutSoc.send(packet);
            // System.out.println("transmitting B frame");
            }
            break;
           default :
               packet = new DatagramPacket(inputBuffer, length, addr, videoOutPort);
```

```
                    videoOutSoc.send(packet);
                    break;
                }
            int stamp2 = (int) System.currentTimeMillis();
    } catch (UnknownHostException e) {
      System.out.println("UnknownHostException: " + e);
    } catch (IOException e) {
      System.out.println("IOException: " + e);
    }
    }
   }
}
}
```

6. BandwidthCheckAgent.java

```
package de.ikv.grasshopper.example;

import de.ikv.grasshopper.example.util.InputDialog;
import de.ikv.grasshopper.agency.MobileAgent;
import de.ikv.grasshopper.type.Location;
import de.ikv.grasshopper.app.util.GOptionPane;
import de.ikv.grasshopper.communication.GrasshopperAddress;
import de.ikv.grasshopper.config.Configurator;

public class BandwidthCheckAgent extends MobileAgent {
        // Location of the remote agency.
        private Location remote = null;

        // Location of the home agency.
        private Location home = null;

        // Free memory of remote agency.
        private long freeBandwidth;

        // The agent's execution state.
        private int state = 0;

        // The input dialog handle
        private transient InputDialog dialogInput;

        // The input dialog status
        private boolean WindowOpened = false;

        // Sets the name of the agent.
        // @return The name of the agent.
        public String getName() {
        return new String("BandwidthCheckAgent");
        }

        // Action, which should be executed, as the agent moves from
        // one place to another place within the same agency

        public void onMove() {
```

```
            System.out.println("onMove called ...");
            if( WindowOpened )
              dialogInput.closeDialog();
    }

// What to do if a user double clicks on the agent entry in the main window see Grasshopper programmer's guide
    public void action() {
            state = 0;
            live();
    }

     // Specifies the agent's behaviour.
    public void live() {

    // at home agency after creation
    if (state == 0) {
            dialogInput = new InputDialog("Destination:    ");
            WindowOpened = true;
            try {
                    String remoteAddress = dialogInput.getInputString();
                    WindowOpened = false;
                            try {
                                    remote = new Location(remoteAddress);
                                    // If there isn't active region
                                    // use configurator to
                                    // get the information about active receiver at the home agency
                                    String [] homeAddress = Configurator.getConfigurator().

    getCommunicationServer().getReceiverAddressesAsString();

                                    // otherwise (if the region is active) just call the getLocation()
                                    // to get active receiver
                                    // home = getLocation();

                                    home = new Location(homeAddress[0]);
                                    System.out.println("home = " + home.toString());
                                    state++;
                                    move(remote);
                    }
                    catch (Exception e) {
                                    System.err.println("BandwidthCheckAgent: " + e.getMessage());
                                    state = 0;
                                    live();
                    }
    }
    catch (Exception e) {
            System.out.println("BandwidthCheckAgent: Exception from InputDialog");
    }
    }

    // arrived at remote agency
    else if (state == 1) {
            System.out.println("Collecting information about free bandwidth ...");
            freeBandwidth = Runtime.getRuntime().freeMemory();
            try {
                            state++;
```

```
                    move(home);
            }
            catch (Exception e) {
                // if region is not running, specify fully qualified address
                // of home agency
                System.out.println("Error: " + e.toString() );
                    InputDialog dialog = new InputDialog("Address of home agency:   ");
                    try {
                        String homeAddress = dialog.getInputString();
                        try {
                            home = new Location(homeAddress);
                            move(home);
                        }
                        catch (Exception e1) {
                            System.err.println("BandwidthCheckAgent: " + e.getMessage());
                        }
                    }
                    catch (Exception e2) {
                    System.out.println("BandwidthCheckAgent: Exception from InputDialog");
                }
            }
        }

        // back home
        else if (state == 2) {
            String[] message = new String[2];
            message[0] = new String("Agency " + remote.getHost() + "/" +
                                                    remote.getAgentsystem() + " has");
            message[1] = new String(freeBandwidth + " bytes of free Bandwidth.");
            GOptionPane.showMessageDialog(null,message,"Free Bandwidth",

        GOptionPane.INFORMATION_MESSAGE);
            state = 0;
            }
        }
}
```