

AN ACTIVE NETWORK BASED APPROACH FOR IMPLEMENTATION OF DIFFERENTIATED SERVICES

A DISSERTATION

*Submitted in partial fulfilment of the
requirements for the award of the degree*

of

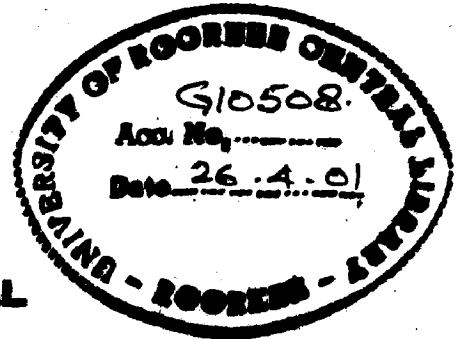
MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND TECHNOLOGY

By

DEEPAK KUMAR GOYAL



**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE-247 667 (INDIA)**

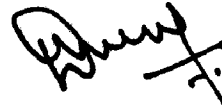
FEBRUARY, 2001

CANDIDATE'S DECLARATION

I hereby declare that the work which is presented in the dissertation
"AN ACTIVE NETWORK BASED APPROACH FOR IMPLEMENTATION
DIFFERENTIATED SERVICES ", in partial fulfillment of the requirements for the
award of the degree of **MASTER OF TECHNOLOGY** with specialization in
COMPUTER SCIENCE AND TECHNOLOGY submitted in the Department of
Electronics & Computer Engineering, University of Roorkee, Roorkee, is an authentic
record of my own original work carried out from July-2000 to February-2001 under the
supervision and guidance of **Dr.(Mrs.) Kumkum Garg, Professor, Department of
Electronics and Computer Engineering, University of Roorkee, Roorkee.**
The matter embodied in this dissertation has not been submitted by me for the award of
any other degree.

Date: 22.02.2001

Place: Roorkee



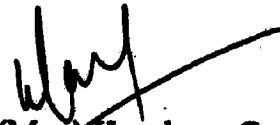
(DEEPAK KUMAR GOYAL)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of
my knowledge and belief.

Date: 22.02.01

Place: Roorkee



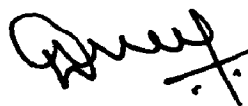
Dr. (Mrs.) Kumkum Garg
Professor E&C Department
University of Roorkee
Roorkee-247667

ACKNOWLEDGEMENT

It is my pleasant duty to thank all who have helped me in completing this dissertation. I take the opportunity to express my deep sense of indebtedness and sincerest gratitude to my guide, Dr. (Mrs.) Kumkum Garg, professor, Electronics and Computer Engineering Department, University of Roorkee, Roorkee for having extended her ever encouraging and affable guidance and constant encouragement throughout the progress of this thesis. She has displayed unique tolerance and understanding at every step of progress and encouraged me incessantly. I deem it my privilege to have carried out the dissertation work under her able guidance.

I am also grateful to Dr. R. P. AGARWAL (Professor & Head of the E&C Dept.,) and [REDACTED] for providing the necessary facilities for the successful completion of this work.

I owe my gratitude to Aashish Suchdeva, Arvind Singh Sengar, Upendra Rajput, Dheeraj Singh and thanks to all my friends for their constant encouragement and morale boosting.

 22/04/2001

(DEEPAK KUMAR GOYAL)

Abstract

A lot of research is being carried out now a days towards an efficient and flexible implementation of different type of services to provide different quality of services as per requirements of the customer. This type of architecture, in which the routing and forwarding functionality accommodates different types of services, is called **Differentiated Services** architecture.

The concept of **Active networks** is also attracting a lot of attention, as a new approach to network architecture. Active networks are more flexible as they incorporate intelligent routers, which can be programmed by the received packets. All routers in an active network are active and can perform packet marking and remarking functions, unlike in existing networks where only edge routers can do so.

In this dissertation, we have proposed a scheme to implement differentiated services over active networks. We show that this scheme gives a better implementation of differentiated services, as compared with existing networks, in terms of bandwidth utilization. The comparison is done by simulating a network and then studying the throughput of different traffic flows at each active node.

The model is developed in JAVA 2 and tested in Win 98 environment.

Contents

	Page no.
1. Introduction	1
1.1 Overview	1
1.2 Motivation for work done	4
1.3 Organization of the report	5
2. Active Networks	6
2.1 Why Active Networking?	6
2.2 Implementation approaches	7
2.2.1 Programmable Switch Approach	7
2.2.2 Capsule Approach	8
2.3 Existing Work in Active Networking	8
2.4 Infrastructure for Active Networks	13
2.5 ANEP (Active Network Encapsulation Protocol)	15
3. Differentiated Services	21
3.1 Differentiated Services Architecture	23
3.1.1 Differentiated Services Architectural Model	25
3.1.2 Differentiated Services Domain	26
3.1.3 Traffic Classification and Conditioning	27
3.2 Location of Traffic Conditioners and MF Classifiers	30
3.2.1 Within the Source Domain	31
3.2.2. At the Boundary of a DS Domain	31
3.2.3. In non-DS-Capable Domains	32

3.2.4. In interior DS Nodes	32
4. Differentiated Services in Active Networks	33
4.1 Packet format	33
4.2 Active Node architecture	35
4.3 Distributed packet marking scheme	36
5. Implementation	39
5.1 Simulation model	39
5.2 Topology	41
5.2.1. Active network	41
5.2.2. Differentiated Services Internet	42
5.3 Implementation detail	44
6. Conclusion	45
6.1 Discussion of results	45
6.2 Suggestions for Further Work	48
6.2.1 Multiple code point implementation	48
6.2.2 New models and algorithms	48
6.2.3 Security issue	48

References

Appendix: Source Listing

Introduction

1.1. Overview

Internet growth in recent times has surpassed all other technologies preceding it, including radio, television and the personal computer. It is expected that the Internet will metamorphose into a medium for the convergence of voice, video and data communications. These will require new services (e.g. guaranteed QoS, congestion avoidance, resource reservation etc.) that cannot be promised by currently existing best effort services model.

The present day Internet is of interconnected routers, which have a fixed, though highly optimized, functionality for routing packets to their destinations. The packets themselves are viewed as pure data. Only service providers who can configure and manage them to provide service can program these routers, otherwise the routers are not programmable. The current proposal to solve this problem is to use Differentiated Services architecture. But this will require very large-scale updates and will also reduce the flexibility towards newer customer requirements [1].

Unfortunately, any change at network level is a very time consuming process. It requires standardization, development, and deployment by vendors and then enabling by administrators. Currently, large-scale deployment of any concept takes years to do so. *Active Networks* will fill the gap between user requirements and deployment of new services to support them, as traditional slow process of consensus based standardization can be avoided [2].

Active Networks offer a new approach where, a network is not just a passive carrier of bits, but a more general computational model. Active networks may be simplistically viewed as a set of active nodes that perform customized operation on the data flowing through them. Unlike the traditional networks, in an active network the packets can contain programs instead of passive data. Such packets are executed at routers or switches in the network and further course of action depends on the outcome of this execution.

The users also see Internet as a closed system, where they have little control over the handling of their data or packets, after it has been put on the Internet. *Active Networks* is an approach where packets contain control information that can affect the network behavior of routers. Thus packets and routing elements can interact in novel ways.

Active Networks will have open programmable routers, on which users would be able to deploy programs dynamically. The functionality of these routers will not be standardized, but their execution environment will be, so that new innovative ideas are not hindered. Besides this, present day networks provide the same service to all the applications, thus removing any possibility for application specific processing.

Active Networks will also help in providing quality of services. As the computing cost decreases, it is possible to add computing within the network, in order to support new emerging applications, or to enhance performance of existing applications.

With programmable routers, not only will existing applications take new shape, but new applications can also be supported. These are unimaginable with the present day Internet. Here we discuss some such applications.

a. Online Auction

In online auctions, a server collects client bids and processes them for each available item. Servers also respond to the requests for the current price. Because of network delay, information responding to such queries may be out of date by the time they reach to the clients. Thus clients may submit a bid much lower than the current price of the item. Servers may have to receive many such bids and reject them. This unnecessary flow of packets simply increases the network traffic and server load and makes whole process slow [3].

In an active network environment, when a server is heavily loaded, it activates the routers to serve as active routers by sending them the current prices of items. Active routers now filter the incoming packets by rejecting low bids before they reach the server. Servers periodically update these routers with the current prices. Since active routers can drop a major part of unnecessary bids, they free up the server resources for processing competitive bids only.

b. Protocol Boosters

This application talks about building protocols dynamically. The idea is to have protocol elements that can be inserted into existing protocols to satisfy some particular application, which could not be supported by the protocol previously. These protocol

elements can be removed from the protocols when not required. As the name suggests, it will help in enhancing the performance of protocols by customizing them according to applications. This can also help in evolving new network technologies [3].

c. Quality of Service

Currently, the Internet provides the same service to all applications, which may have diverse quality of service expectations. The requirement is to extend the network architecture to allow service differentiation mechanisms so that some applications can get better services than others.

Active networks can be used to provide this service differentiation. Active routers can be used to evaluate the packets and can provide different kinds of services depending on origin or destination of packets or priority of one type of packets over others. Active routers can dynamically reroute the packets along different paths in order to provide less propagation delay and less packet loss rate to higher priority packets [4].

1.2 Motivation for work done

The present DS architecture requires large implementations to meet the projected demands of the next generation networks. As said above, active networks can provide a convenient and faster alternative. This dissertation work is done with an aim to incorporate this concept in the Differentiated services model so as to optimize the resource utilization in the network.

1.3 Organization of the report

Chapter 2 deals with general concepts related to active networking. It discusses about motivations and applications of active networking. It gives an overview of the various research activities going on in the field of active networking. This is followed by a description of the infrastructure required for active networking and a protocol proposed by **Active Network Group** (*a group comprising of prominent researchers in the field of Active Networking*) called ANEP (Active Network Encapsulation Protocol).

Chapter 3 gives information about Differentiated services. It briefly discusses the concept and the requirement of Differentiated services. An overview of the Differentiated services architecture, which is being implemented in existing Internet, is also given.

Chapter 4 discusses an Active Networks based approach to implement the differentiated services. We will also see how this approach is different from the existing Differentiated services and how they help in getting better performance from the network.

Chapter 5 gives details of simulation model used. It gives the description of used topology and other assumptions, which are used for experiment. Last section of this chapter gives implementation detail about the model of Active Network that is use for this dissertation.

Chapter 6 concludes the dissertation with discussion of results, which makes graphical comparison between new and existing approaches. The section gives suggestions for future work.

Active Networks

2.1. Why Active Networking?

There are many new emerging applications, which require computing in the network. These applications include web proxies, on-line auctions, firewalls, videoconferencing etc. In the present model of the Internet we cannot change protocols easily to satisfy the different service requirements of such applications as hardware and software are bundled together. Thus different kinds of adhoc approaches are being used to fulfill the different service requirements [5].

The goal is to replace all these adhoc approaches with a generic programming capability of the network. Active networks can change the structure of networks by decoupling the hardware and software. They will allow new services to be deployed at a faster pace to suit the applications.

Major motivations for active networking can be listed as:

- To minimize the requirements of standard protocols to develop end-to-end services and to support dynamic modification of network behavior.
- To provide mechanisms for supporting different levels/qualities/classes of services.
- To maximize flexibility in service by supporting different services simultaneously.
- To support network management at all levels.
- To support easy and fast deployment of new protocols.

2.2 Implementation approaches

Today, two approaches are being used for implementing active networks [6]. Each of these approaches defines a different way of injecting programs in the network.

2.2.1 Programmable Switch Approach

In this approach, packets are of two categories: packets containing programs only and simple data packets. This approach separates injection of these two types of packets. The routers of the network are programmable switches [6]. Injecting program packets first programs the routers, then data packets are sent on the network. The routers then work on these data packets as dictated by the programs that have been loaded on to the various routers.

In essence, this approach preserves the present structure of network by distinguishing between data transfer and network management. This approach is well suited for network administrators to control the network. It could also be useful in preventing malicious programming of networks, because the people who may program the network should be authorized.

The problem with this approach is that it does not provide full flexibility as promised by active networking. The other problem is that, in general, we do not know which path is going to be followed by a particular packet for a certain destination. Packets originating from a common source and destined for a common destination may take different paths, and it is not possible always to program all routers between source and destination.

2.2.2 Capsule Approach

In this approach, every packet, or capsule, contains a program. Data may also be embedded within the program. When such packets arrive at a router, the program is executed and the next action depends on the outcome of program execution. The programs are of basic instructions that may operate on their data or may also program the router itself [6].

The problems with this approach are of security and efficiency. A malicious user can harm the whole network by programming the router. Besides since each packet has a program that is executed at each router so the whole network is slow. Steps are being taken to solve these problems. The security problem can be solved by restricting the address space, which a packet can access and the performance problem can be solved by making programs small and of basic instruction, which are stored on the router.

This approach solves the problem with previous approach of finding routers to program, in a certain path. The capsule approach also provides full flexibility to the end-users for providing customized processing on their data.

2.3 Existing Work in Active Networking

We now survey existing work on active networking that is currently ongoing at a number of research institutes. The major areas of research are studying capsule and programmable switch approaches, developing applications, languages and compilers for active networks.

Massachusetts Institute of Technology

At MIT, researchers have developed an Active Network Tool Kit, ANTS (Active Network Transfer System)[7]. ANTS is based on the capsule approach. It uses mobile code for deployment of new protocols at both end systems and intermediate routers. The aim of ANTS is to develop a standardized communication model that will support many protocols simultaneously.

New protocols are introduced by specifying the routines to be executed at the routers on the forwarded packets. The routines will be deployed online, thus avoiding an off-line process of reconfiguring a router. In ANTS, communication is done by exchanging capsules, rather than simple data packets. A router in ANTS is an active router on which capsules are processed. New network services in ANTS are created by defining new capsule types and their processing. There are mainly three components in ANTS Capsules, Active routers and Code distribution.

Capsule

In ANTS, a capsule is the entity which is sent between active routers. Unlike traditional packets, capsules contain application data and describe the processing they require within the network.

A new service is developed by related types of capsules, to create a protocol. The processing, which a capsule will require at a router, is kept as a reference to a forwarding routine in each capsule. Capsules belonging to the same protocol can communicate with each other through shared variables available at active routers, while capsules belonging

to different protocols can't communicate. This is provided as a protection measure by active routers.

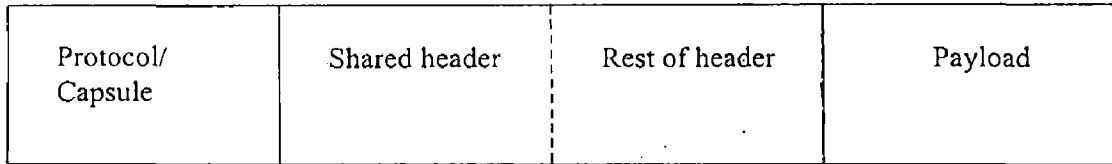


Fig 2.1. ANTS packet format

An ANTS capsule format is given in Figure 2.1. The **protocol** field is an identifier for the protocol and forwarding routine within that protocol. This identifier is based on a secure hash of the forwarding routines in the protocol, which provides a fingerprint of the code. Since it depends only on the fingerprint, thus no central authority is needed to assign identifiers and also an active router can check it independently.

The rest of the packet consists of a common header present in all packets; a type checked header that may be modified as the packet traverses the network and the payload.

Active routers

In ANTS, active routers provide the environment in which protocols are executed. Active routers also restrict the access of executing programs to shared resources. They also provide a set of primitives that are used to construct forwarding routines. Currently following router primitives are available.

Environment primitives: These primitives give information about the router environment like router location, state of links and routing tables [7].

Storage primitives: These primitives help in storage of application-defined objects, including other capsules. The objects, which are stored, are not permanent. The

primitives also help in deleting the old primitives so the network does not retain stale information.

Control primitives: These primitives allow capsules to create other capsules and forward, suspend and discard themselves. They are also used to route the capsules.

Capsule manipulation: These primitives are used to manipulate capsule header and payloads.

Active router uses mobile code techniques to execute unauthenticated routines in restricted environments. Active routers also limit resources that can be consumed by a packet. For this, capsules carry a resource limit that is decremented by routers when resources are consumed and a packet is discarded when it reaches zero.

Code distribution

The third component of ANTS architecture sends forwarding routines to the router where they are required. Applications provide local routers the necessary forwarding routines, which they will need to process capsules. These forwarding routines are transferred using a lightweight protocol along the path that a capsule follows. When a capsule reaches a router, which does not have the forwarding routine required for that capsule type, the capsule is suspended.

The router then asks the forwarding routine from the previous router of the path the capsule has followed. If the requested router has forwarding routines, then it immediately sends them to the requesting router, else the request is propagated.

All this is done in real time, so when a specified time expires and the router does not get required forwarding routine, all the capsules belonging to that protocol is discarded. This approach where a router relies upon previous router, creates a region in which every

router will soon have forwarding routines so code transfer is eventually not needed which speeds up execution.

Currently at MIT research is focusing on new services that can be introduced to active networks and how these services are useful for improving overall application performance [7].

University of Columbia

The NetScript project at Columbia University consists of a programming language and execution environment. The aim of project is to deploy new protocols and services in the network to simplify the development of networks. The NetScript programming language provides the means to script processing of packet streams. In NetScript, programs that can be dynamically dispatched to routers are called agents.

Agents are executed at the remote hosts and are used to control the functions of intermediate routers and for program management. An agent binds primitive router functions with processing packet streams and allocates router resources. Agents can be programmed to handle user-defined processing along with standardize protocols. Every incoming packet stream is allocated to an appropriate agent deployed on the router that performs the desired functionality of the protocol [8].

University of Pennsylvania

The Switchware project at UPenn proposes a programmable switch that allows type checked modules to be loaded in routers of the network. The switch consists of input and output ports controlled by software programmable elements. Active packets are sent to inject the program in the network, and the switchware routers execute the code fragment

in each packet along its delivery path. The delivery path is controlled by the mobile code itself.

A scripting language PLAN (Packet Language for Active Networks) has also been developed for programming the interface of routers. Every active packet in switchware carries PLAN program. PLAN is based on the lambda calculus [8].

Other research activities

At Georgia Institute of Technology, attempts are being made to develop trustworthy services using active network concepts. They have developed effective congestion control mechanisms by allowing applications to invoke special-purpose algorithms at the time of congestion. *At Bell Communications Research*, efforts are to specify semantics for active routers and investing these semantics in collaboration with the UPenn Team.

At Carnegie Mellon University, efforts are towards developing resource management mechanisms for supporting application-aware networks. They are also exploring support for sophisticated multi-party applications like video-conferencing.

At University of Arizona, researchers in the Liquid Software Project are exploring mechanisms for building networked systems using liquid software. Liquid software refers to software, which can easily flow from machine to machines. Their main focus is towards fast compilation of mobile code, mobile search applications and OS support.

2.4 Infrastructure for Active Networks

As we know, a packet traverses through a number of routers during its communication path from source to destination, so the program, that a packet may contain, will be

executed on variety of platforms. Thus, to satisfy the needs of these varieties of platforms, a common framework has been proposed. It suggests a common programming model for program encoding, router resources and services available at a router.

According to the model, the program should be able to execute on a variety of platforms. This objective is achieved by choosing a platform-independent language like Java [9] for program encoding. The other requirement for programs is of security i.e. the address space a program can access on a router should be restricted.

Besides the above two requirements, execution of a program on a router should not degrade network performance, so the program should be small and be composed of basic instructions to facilitate rapid execution. These basic instructions, or services, could be made available at a router dynamically. The services could do basic operations of packet manipulation such as changing data or header. The services could also provide basic functionalities to control flow.

The router resources, which a router can access, should also be standardized. The router resources could be logical like routing table as well as physical like processing capacity, storage and bandwidth etc.

The major issue with active networking is about its inter-operability with present networks. Efforts are being made to use existing infrastructure without a complete overhaul. The idea is to use *tunneling*. The active routers would tunnel their packets through the legacy routers as done in the case of MBONE, where multicasting routers use tunneling to pass their packets through non-multicasting routers.

To support this idea an extension to IP protocol, *Active IP* [10], has been suggested. The extended IP protocol would use option field of normal IP packet for embedding program fragment in the IP datagram.

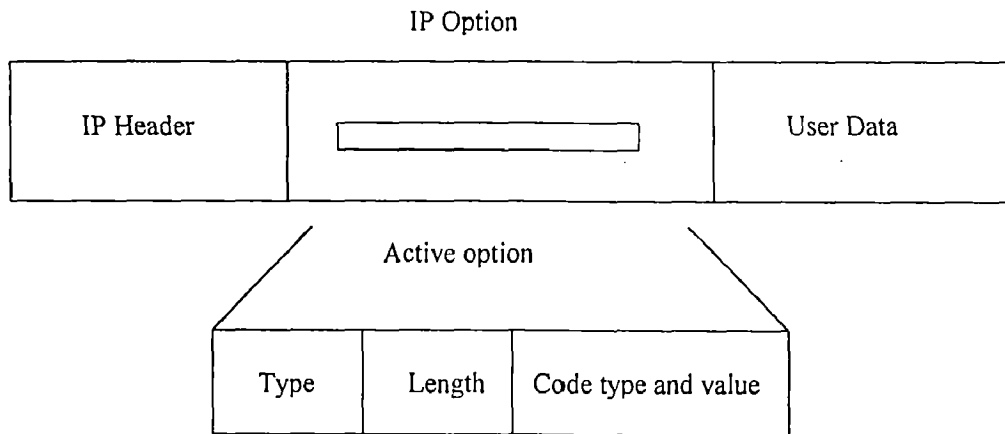


Fig 2.2.Active IP Packet Format

As we know, legacy routers do not look at the option field, they will just pass the packet as a simple IP packet, while active routers would execute the code fragment.

2.5 ANEP (Active Network Encapsulation Protocol)

An active network router is capable of dynamically loading and executing programs written in a variety of languages. These programs are carried in the payload of an active network frame. The program is executed by a receiving router in the environment specified by the ANEP [11]. Various options can be specified in the ANEP header, such as authentication, confidentiality, or integrity.

Next, we describe the syntax and semantics of ANEP. The details of handling the contents of an active frame are left to the individual implementations/environments.

The reasons an active network header is necessary are:

- a) An active router receiving a packet must be able to uniquely and quickly determine the environment in which it is intended to be evaluated.
- b) To allow minimal, default processing of packets for which the intended evaluation environment is unavailable.
- c) 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 format of the ANEP header [11] is given in fig 2.3.

Version	Flags	Type ID
ANEP Header Length		ANEP Packet Length
Options		
Payload		

Fig 2.3 ANEP packet

All fields larger than one octet must be in network-byte order (big endian format). This holds for all Options as well.

The *Version* field indicates the header format in use. This description is for version 1. This field will be changed if the ANEP header should change. If an active router receives a packet whose version number it does not recognize, it should discard the packet. The length of this field is 8 bits.

The *Flags* field is 8 bits long. In version 1 of this protocol, only the most significant bit is used, to indicate what the router should do if it does not recognize the Type ID. If the

value is 0, the router could try to forward the packet using the default routing mechanism (if one is in use), if the necessary information is available in the Options part of the header. If the value is 1, the router should discard the packet. The rest of the bits in this field should be ignored by the router. It is recommended that they be set to zero by the packet originator.

The *ANEP Header Length* field specifies the length of the ANEP header in 32 bit words. If no options are included in the packet, then its value must be 2. The length of this field is 16 bits.

The *Type ID* field indicates the evaluation environment of the message. The active router should evaluate the packet in the proper environment. The length of this field is 16 bits.

The proper authority for assigning Type ID values to interested parties is the Active Networks Assigned Numbers Authority (ANANA). The Type ID value 0 is reserved for possible future network layer informational and error messages. If the value contained in this field is not recognized, the router 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, including the packet payload, in octets. This field is used to recover the packet if it has been transmitted over a lower layer that does not allow recovery of the packet length. The length of this field is 16 bits. Notice, that unlike other length fields in this document, the unit of measure is octets.

Options in the form of TLVs (Type/Length/Value) can be included in the packet, immediately following the basic header. The format of these options given in fig 2.4.

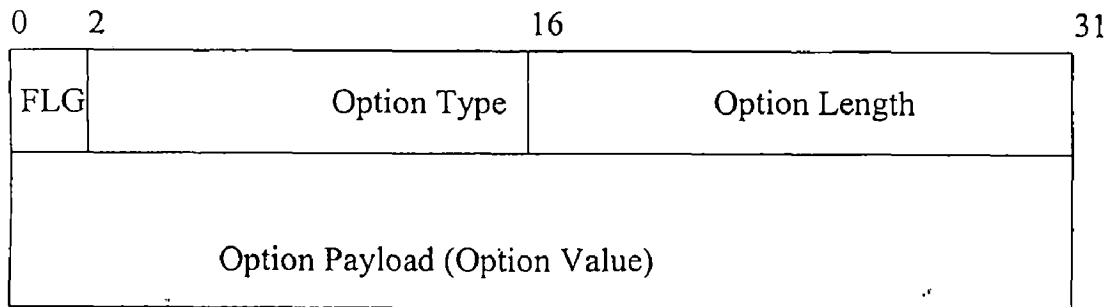


Fig 2.4. Format of ANEP option

The Option Type field identifies the option. How the active router handles the Option Payload depends on the Option Type value. The length of this field is 14 bits. The following values have been reserved:

- Source Identifier 1
- Destination Identifier 2
- Integrity Checksum 3
- N/N authentication 4

All values intended for public use are under the authority of the Active Networks Assigned Numbers Authority (ANANA). Other parties can use their own values for this field if the most significant bit (Flags bit 0) is set. These Options are only meaningful inside the specified evaluation environment, so the proper authority for assigning these values is the Type ID owner.

The Option Length field contains the length of the TLV in 32 bit words. This includes the length of the Flags, Option Type, and Option Length and Option Payload fields. This value must never be less than 1 (for an option with a zero sized Option payload). If the Option payload size is larger than the size of the data it carries, it is recommended that the excess 1 - 3 octets be zero filled and be ignored by a receiving implementation. The length of this field is 16 bits.

The two most significant bits of the first word in an Option (bits 0 and 1) are used as a Flag field. Bit 0 (Private) is used to indicate that the Option Type is only meaningful within the specified Type ID. The router should not try to parse the Option at packet receipt if this bit is set. If the active router does not know how to process the indicated Option Type, the action taken is defined by the value of bit 1 of the Flags field. If this is 0 this option is ignored and processing the header is continued. If it is 1 the packet is discarded.

Defined Options

This section briefly discusses the options defined above.

Source Identifier

This Option includes a value, which uniquely identifies the sender of the packet within the active network. The payload of this Option consists of a 32-bit value, which identifies the addressing scheme in use, followed by that scheme's data.

Destination Identifier

This Option includes a value which uniquely identifies an ultimate destination of the packet within the active network. The format of the payload of this Option is the same as that of the Source Identifier TLV. This field could be used by active routers on which the

intended evaluation environment is unavailable in order to attempt to forward the packet towards an active router capable of better handling the packet.

Integrity Checksum

The payload of this Option contains the 16 bits one's complement of the one's complement sum of the entire ANEP packet, starting with the ANEP Version field. For computing the checksum, the payload of this Option must be set to zero. The Option Length field must be 2.

Non-Negotiated Authentication

This option is used to provide one-way authentication, with no prior negotiation between the packet originator and processing router(s). The payload of this Option consists of a 32-bit value, which identifies the authentication scheme in use, followed by that scheme's data.

It is expected that this option will be used when the number of packets that require authentication is too small to justify the cost of a full negotiation, when the operation is time critical, or when security negotiation cannot take place. The processing cost of this option is expected to be higher than that of a negotiated authentication option, and it might not provide guarantees as hard as the latter, especially with respect to replay protection.

Differentiated Services

The transformation of the Internet into an important and ubiquitous commercial infrastructure has not only created rapidly rising bandwidth demand but also significantly changed consumer expectations in terms of performance, security, and services [4]. Consequently, service providers need to not only evolve their networks to higher and higher speeds but also need to plan for the introduction of services of increasing sophistication, so as to address the varied requirements of different customers. At the same time, Internet Service Providers (ISPs) would like to maximize the sharing of the costly backbone infrastructure in a manner that enables them to control usage of network resources in accordance with service pricing and revenue potential.

The two trends of rapidly rising bandwidth demand and rising need for differentiation has resulted in intense efforts to build fast packet forwarding engines and in efforts to define mechanisms for service differentiation. Isolating traffic from different customers and providing minimum bandwidth guarantees in a customer specific manner, allows customers of ISP services to determine the bandwidth they require to satisfy their needs based on their own traffic requirements, just as they would with a leased line. They may want the additional flexibility of being able to specify the manner in which their internal traffic, from different sources, is allowed access to the available bandwidth.

Furthermore, it may be desirable to define different levels of service [1] for different types of traffic in a customer dependent manner. For example, some customers may consider FTP or Web transfers to be low priority, and so for them the service provider could aggregate multiple flows of these types. However, other customers may define Voice-over-IP or database queries to be high priority, and therefore for them the service provider must ensure good performance by giving these traffic types priority or guaranteed minimum bandwidth from within that customer's available bandwidth.

In another scenario, some customers may require extremely reliable and predictable performance for a small set of applications. They may indicate this requirement to the network as dynamic reservations of exact bandwidth along with specification of delay bounds. This indication may be by explicit signaling using a resource reservation protocol like RSVP, or may be done implicitly by some other means. The service provider's infrastructure must be capable of allowing end-users to choose such a stringent, although possibly expensive, service if they require it.

On the other hand, service providers want to maximize the sharing and multiplexing of their infrastructure, service providers should have the flexibility to distinguish themselves from other service providers by being able to tailor their service offerings in whatever competitive manner they choose to do.

To meet the above requirements, the Network Working Group at IETF has proposed an architecture called the DS-Architecture. The following sub sections discuss it.

3.1. Differentiated Services Architecture

This architecture is composed of a number of functional elements implemented in network nodes, including a small set of per-hop forwarding behaviors, packet classification functions, and traffic conditioning functions including metering, marking, shaping, and policing.

Here scalability is achieved by implementing complex classification and conditioning functions only at network boundary nodes, and by applying per-hop behaviors to aggregates of traffic, which have been appropriately marked using the DS field in the IPv4 or IPv6 headers [1]. Per-hop behaviors are defined to permit a reasonably granular means of allocating buffer and bandwidth resources at each node among competing traffic streams

This architecture is aimed at distinguishing the following:

- The service provided to a traffic aggregate,
- The conditioning functions and per-hop behaviors used to realize services,
- The DS field value (DS codepoint) used to mark packets to select a per-hop behavior, and
- The particular node implementation mechanisms, which realize a per-hop behavior.

Service provisioning and traffic conditioning policies are sufficiently decoupled from the forwarding behaviors within the network interior to permit implementation of a wide variety of service behaviors, with room for future expansion.

This architecture only provides service differentiation in one direction of traffic flow and is therefore asymmetric.

Next we give a general conceptual overview of the terms used in DS architecture :

DS behavior aggregate: - A collection of packets with the same DS codepoint crosses a link in a particular direction.

DS codepoint: - A specific value of the DSCP portion of the DS field, used to select a PHB (Per Hop Behavior).

DS field: - The IPv4 header TOS octet or the IPv6 Traffic Class octet when interpreted in conformance with the definition given in [1]. The bits of the DSCP field encode the DS codepoint, while the remaining bits are currently unused.

DS region: - A set of contiguous DS domains, which can offer, differentiated services over paths across those DS domains.

Per-Hop-Behavior (PHB): - The externally observable forwarding behavior applied at a DS-compliant node to a DS behavior aggregate.

Service Level Agreement (SLA): - A service contract between a customer and a service provider that specifies the forwarding service a customer should receive. A customer may be a user organization (source domain) or another DS domain (upstream domain). A SLA may include traffic conditioning rules, which constitute a TCA in whole or in part.

Source domain: - A domain, which contains the node(s), originating the traffic receiving a particular service.

Traffic conditioner: - An entity, which performs traffic conditioning functions and which may contain meters, markers, droppers, and shapers. Traffic conditioners are typically deployed in DS boundary nodes only. A traffic conditioner may re-mark a traffic

stream or may discard or shape packets to alter the temporal characteristics of the stream and bring it into compliance with a traffic profile.

Traffic conditioning: - Control functions performed to enforce rules specified in a TCA, including metering, marking, shaping, and policing.

Traffic Conditioning Agreement: - An agreement specifying classifier rules Agreement (TCA) and any corresponding traffic profiles and metering, marking, discarding and/or shaping rules which are to apply to the traffic streams selected by the classifier. A TCA encompasses all of the traffic conditioning rules explicitly specified within a SLA along with all of the rules implicit from the relevant service requirements and/or from a DS domain's service provisioning policy.

Traffic profile: - A description of the temporal properties of a traffic stream such as rate and burst size.

Traffic stream: - An administratively significant set of one or more microflows, which traverse a path, segment. A traffic stream may consist of the set of active microflows, which are selected by a particular classifier.

3.1.1 Differentiated Services Architectural Model

The differentiated services architecture is based on a simple model where traffic entering a network is classified and possibly conditioned at the boundaries of the network, and assigned to different behavior aggregates. Each behavior aggregate is identified by a single DS codepoint. Within the core of the network, packets are forwarded according to the per-hop behavior associated with the DS codepoint. In this section, we discuss the

key components within a differentiated services region, traffic classification and conditioning functions, and how differentiated services are achieved through the combination of traffic conditioning and PHB-based forwarding.

3.1.2. Differentiated Services Domain and Differentiated Services Region

A DS domain is a contiguous set of DS nodes, which operate with a common service provisioning policy and set of PHB groups implemented on each node. A DS domain has a well-defined boundary consisting of DS boundary nodes, which classify and possibly condition ingress traffic to ensure that packets which transit the domain are appropriately marked to select a PHB from one of the PHB groups supported within the domain. Nodes within the DS domain select the forwarding behavior for packets based on their DS codepoint, mapping that value to one of the supported PHBs using either the recommended codepoint->PHB mapping or a locally customized mapping.

a. DS Boundary Nodes and Interior Nodes

A DS domain consists of DS boundary nodes and DS interior nodes. DS boundary nodes interconnect the DS domain to other DS or non-DS-capable domains, whilst DS interior nodes only connect to other DS interior or boundary nodes within the same DS domain.

Both DS boundary nodes and interior nodes must be able to apply the appropriate PHB to packets based on the DS codepoint; otherwise unpredictable behavior may result. In addition, DS boundary nodes may be required to perform traffic conditioning functions as

defined by a traffic conditioning agreement (TCA) between their DS domain and the peering domain.

Interior nodes may be able to perform limited traffic conditioning functions such as DS codepoint re-marking. Interior nodes, which implement more complex classification and traffic conditioning functions, are analogous to DS boundary nodes.

b. DS Ingress Node and Egress Node

DS boundary nodes act both as a DS ingress node and as a DS egress node for different directions of traffic. Traffic enters a DS domain at a DS ingress node and leaves a DS domain at a DS egress node. A DS ingress node is responsible for ensuring that the traffic entering the DS domain conforms to any TCA between it and the other domain to which the ingress node is connected.

c. Differentiated Services Region

A differentiated services region (DS Region) is a set of one or more contiguous DS domains. DS regions are capable of supporting differentiated services along paths which span the domains within the region.

3.1.3 Traffic Classification and Conditioning

Differentiated services are extended across a DS domain boundary by establishing a SLA (Service Level Agreement) between an upstream network and a downstream DS domain.

The SLA may specify packet classification and re-marking rules and may also specify traffic profiles and actions to traffic streams, which are in- or out-of-profile. The TCA between the domains is derived (explicitly or implicitly) from this SLA.

The packet classification policy identifies the subset of traffic, which may receive a differentiated service by being conditioned and, or mapped to one or more behavior aggregates (by DS codepoint re-marking) within the DS domain.

Traffic conditioning performs metering, shaping, policing and/or re-marking to ensure that the traffic entering the DS domain conforms to the rules specified in the TCA, in accordance with the domain's service provisioning policy. The extent of traffic conditioning required is dependent on the specifics of the service offering, and may range from simple codepoint re-marking to complex policing and shaping operations.

a. Classifiers

Packet classifiers select packets in a traffic stream based on the content of some portion of the packet header. We define two types of classifiers. The BA (Behavior Aggregate) Classifier classifies packets based on the DS codepoint only. The MF (Multi-Field) classifier selects packets based on the value of a combination of one or more header fields, such as source address, destination address, DS field, protocol ID, source port and destination port numbers, and other information such as incoming interface.

b. Traffic Conditioners

A traffic conditioner may contain the following elements: meter, marker, shaper, and dropper. A traffic stream is selected by a classifier, which steers the packets to a logical instance of a traffic conditioner. A meter is used (where appropriate) to measure the traffic stream against a traffic profile. The state of the meter with respect to a particular packet (e.g., whether it is in- or out-of-profile) may be used to affect a marking, dropping, or shaping action.

When packets exit the traffic conditioner of a DS boundary node the DS codepoint of each packet must be set to an appropriate value.

Fig.3.1 shows the block diagram of a classifier and traffic conditioner. Note that a traffic conditioner may not necessarily contain all four elements. For example, in the case where no traffic profile is in effect,

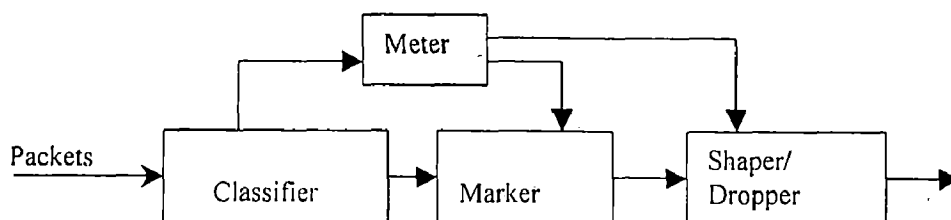


Fig.3.1. Logical View of a Packet Classifier and Traffic Conditioner

Meters

Traffic meters measure the temporal properties of the stream of packets selected by a classifier against a traffic profile specified in a TCA. A meter passes state information to other conditioning functions to trigger a particular action for each packet which is either in- or out-of-profile (to some extent).

Markers

Packet markers set the DS field of a packet to a particular codepoint, adding the marked packet to a particular DS behavior aggregate. The marker may be configured to mark all packets which are steered to it to a single codepoint, or may be configured to mark a packet to one of a set of codepoint used to select a PHB in a PHB group, according to the state of a meter. When the marker changes the codepoint in a packet it is said to have "re-marked" the packet.

Shapers

Shapers delay some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. A shaper usually has a finite-size buffer, and packets may be discarded if there is not sufficient buffer space to hold the delayed packets.

Droppers

Droppers discard some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. This process is known as "policing" the stream. Note that a dropper can be implemented as a special case of a shaper by setting the shaper buffer size to zero (or a few) packets.

3.2 Location of Traffic Conditioners and MF Classifiers

Traffic conditioners are usually located within DS ingress and egress boundary nodes, but may also be located in nodes within the interior of a DS domain, or within a non-DS-capable domain [1].

3.2.1 Within the Source Domain

Source domain is the domain containing the node(s), which originate the traffic receiving a particular service. The traffic originating from the source domain across a boundary may be marked by the traffic sources directly or by intermediate nodes before leaving the source domain. This is referred to as initial marking or "pre-marking".

There are some advantages to marking packets close to the traffic source. First, a traffic source can more easily take an application's preferences into account when deciding which packets

should receive better forwarding treatment. Also, classification of packets is much simpler before the traffic has been aggregated with packets from other sources, since the number of classification rules, which need to be applied within a single node, is reduced.

3.2.2 At the Boundary of a DS Domain

Traffic streams may be classified, marked, and otherwise conditioned on either end of a boundary link (the DS egress node of the upstream domain or the DS ingress node of the downstream domain). When packets are pre-marked and conditioned in the upstream domain, potentially fewer classification and traffic conditioning rules need to be supported in the downstream DS domain. In this case the downstream DS domain may only need to re-mark or police the incoming behavior aggregates to enforce the TCA.

3.2.3 In non-DS-Capable Domains

Traffic sources or intermediate nodes in a non-DS-capable domain may employ traffic conditioners to pre-mark traffic before it reaches the ingress of a downstream DS domain. In this way the local policies for classification and marking may be concealed.

3.2.4 In Interior DS Nodes

The DS architecture allows the implementation of complex classification and traffic conditioning functions in the interior of the network. This approach may have scaling limits, due to the potentially large number of classification and conditioning rules that might need to be maintained.

Differentiated Services in Active Networks

As described in the last chapter, functions like marking, policing, traffic conditioning are performed at the edge of the network. The core router of existing Internet is not allowed to make any changes in the marking policies [4]. They just forward the packets according to a pre-programmed forwarding behavior.

On the other hand, Active routers have no such type of constraint. They can make any changes in the contents of the packet. So the traffic policing functions can be programmed at any router in a very dynamic and flexible manner. Active routers also share information regarding network state (e.g. presence and location of congestion) with other routers. An active router can also ask its neighboring active routers about an alternative path to avoid congestion or to provide better quality of service.

In this chapter we describe the modified packet format and active node architecture that support differentiated services. We also discuss a distributed packet-marking scheme that is meant to optimize network resource utilization.

4.1 Packet format

The option field of ANEP [11] header has been modified to support differentiated services.

Fig 4.1 shows the packet format of ANEP.

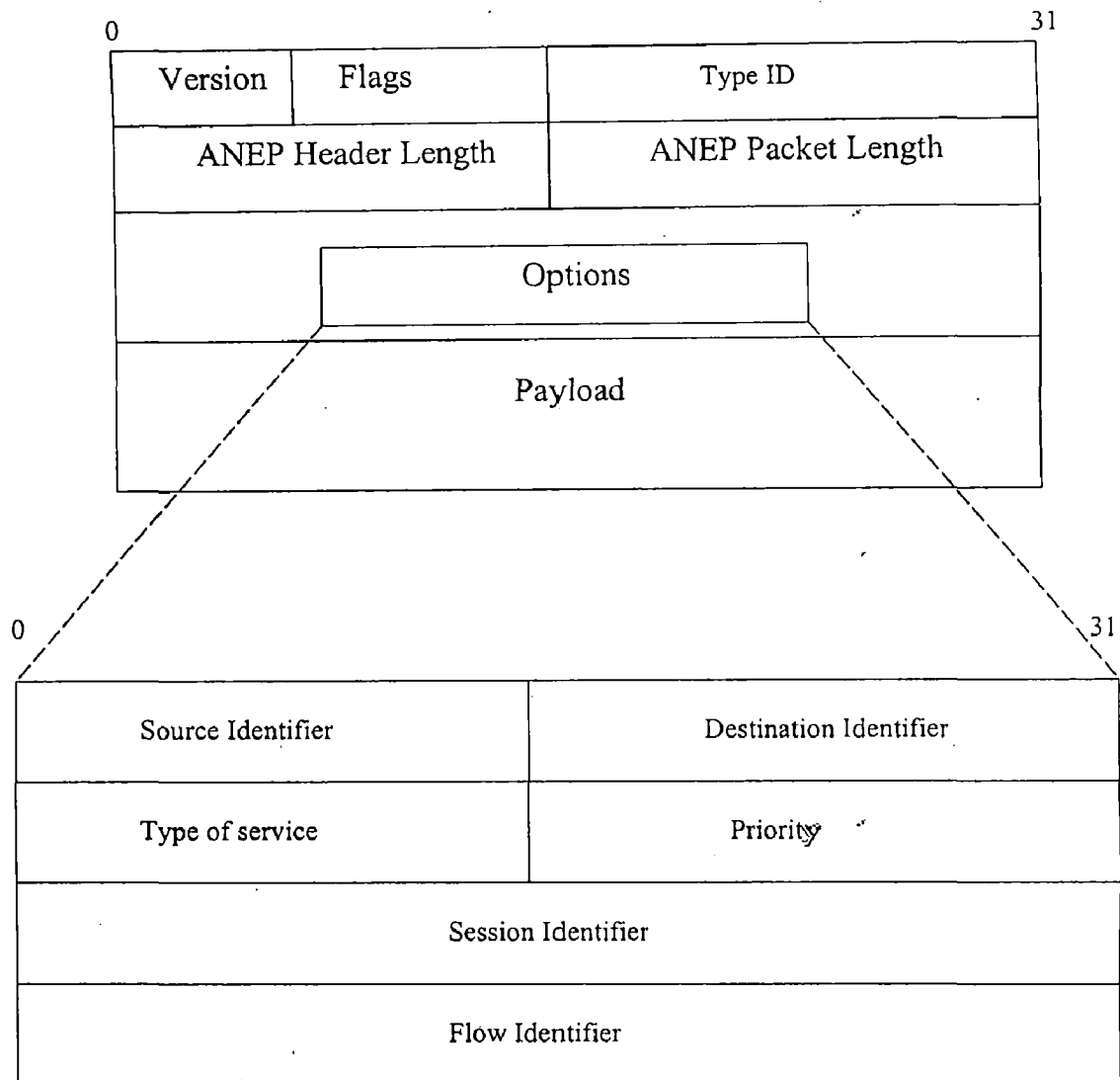


Fig 4.1. ANEP packet Format

All fields except option field are same as described in section 2.6.

Source Identifier and *destination Identifier* can be IP address of source and destination respectively.

Using *Type of Service* field, active routers identify the type of *service agreement* that network have with particular flow or its packet.

The *priority* field defines the relative preference of this particular flow in comparison with other flows in the network. This field may change at any node through out the propagation of a packet from source to destination. In our experiment we use multiple

queues of different priority. On the basis of *priority* field, router decides in which queue the packet should go.

The *Session Identifier* field is used to identify the specific session in which the Active packet will execute. The Flow identifier identifies different flows from same source or different sources.

4.2 Active node architecture

In this dissertation an active node architecture shown in Fig. 4.2 is used. The various components of this architecture are discussed below.

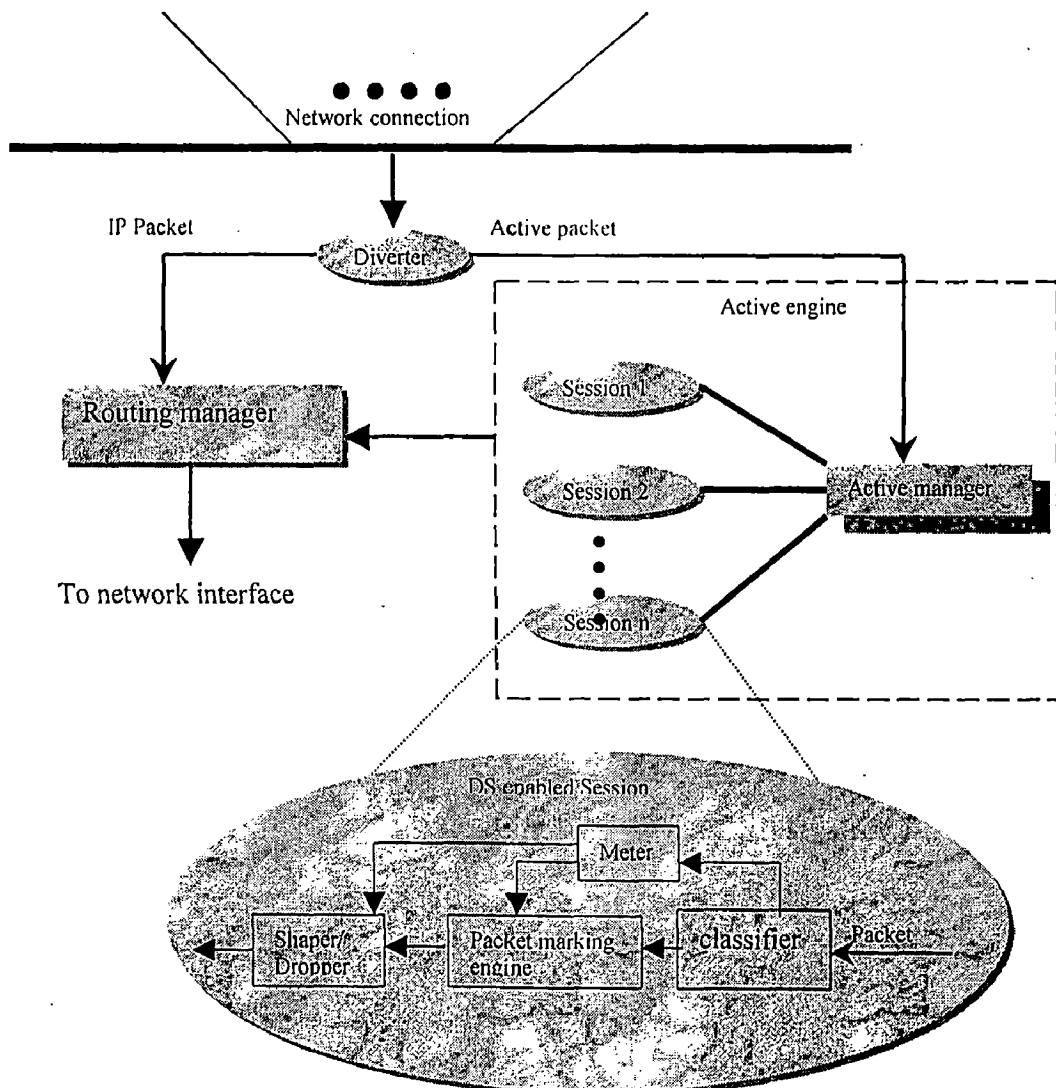


Fig. 4.2. Active router architecture for differentiated services

Diverter - a part of the router that enables it to divert the packets to active engine (AE) based on their header. If it finds any IP packet, it directs it to routing manager. The new generations of high performance IP routers have this option implemented as part of their hardware [12].

Active manager – the core of the active engine is active manager. This module generates the session, co-ordinates data transfer to and from the session, and cleans up after a session when it terminates. While a session is alive, the active manager monitors session resource uses, and can decide to terminate its operation, if it consumes too much resources (CPU time, bandwidth) or tries to violate its action permission.

The work of the **meter, marker, and shaper** [1] is the same as described in section 3.1.3.

4.3 Distributed packet marking scheme

The Differentiated services Architecture gives high priority and provides high reliability to marked traffic. But this marked traffic may degrade other flows, which share the path (or a segment of it) with the high priority flow. As shown in Fig. 4.3, there are three different flows passing through the DS domain.

The one from S1 to R1 is the flow with a high priority, the others, from S2 to R2 and from S3 to R3 are the ones with low priority. Router n9 has congestion on the outgoing link to router n10. So the requirement is that the high priority packets should not be lost because of congestion and they should be forwarded on preference.

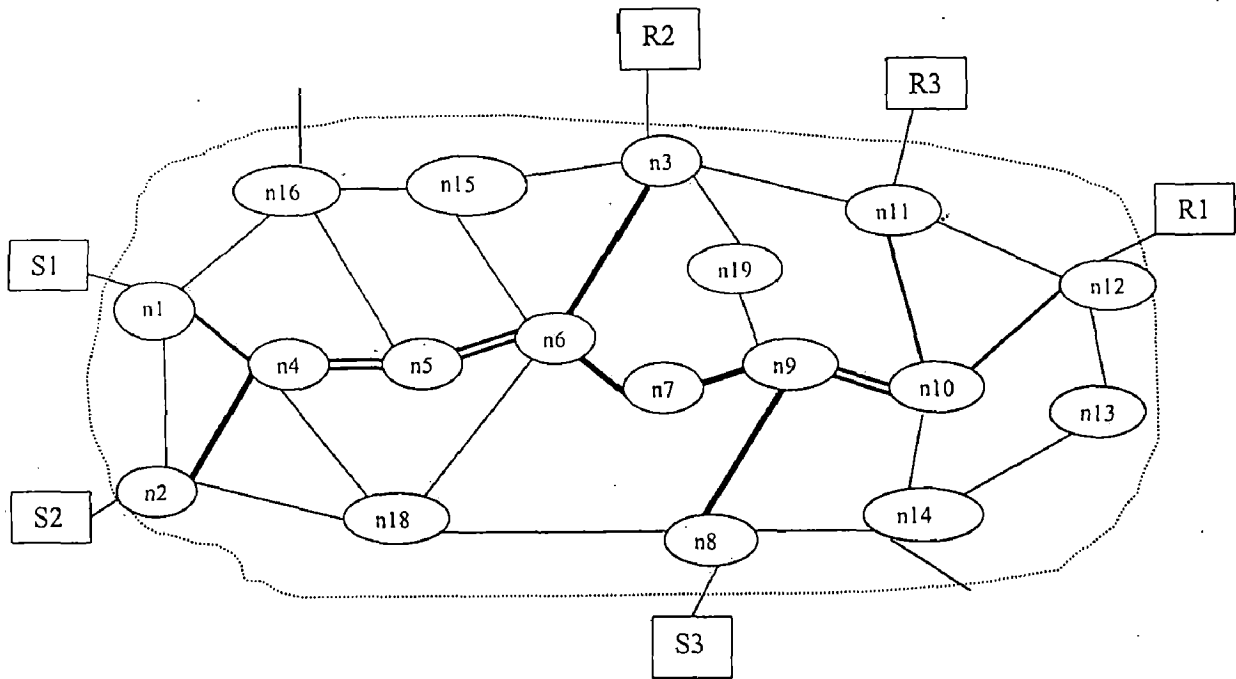


Fig 4.3 Active network Topology

In the present architecture of the Internet, for providing this service, we have to mark packets at router n1. This will result in preferred forwarding behavior for the flow along the path. But this may degrade the performance of the flow from S2 to R2. This degradation depends on the forwarding behavior of router and different type of flows, which are passing through the router. In Active Networks, the packet marking can be done at any router. So, if packet marking is done at the router preceding the congested router, we can improve the throughput of other flows, which share a segment of the path with the high priority flow and which do not pass through the congested router.

Thus, if we mark the packets at router n7, service degradation of flow S2-R2 can be avoided. Before marking at n7, these packets will be considered as normal packets. When we mark packets at n7, router n9 will give them high priority and pass them with preference. At router n10, we again mark them as normal packets. Further down the path, these packets can be marked as per requirement.

Implementation

This chapter describes the simulation model that we have used for our experiment. This will be followed by a description of various software routines and entities used to simulate the network behavior. This chapter also describes the topology of the network simulated and the graphical user interface that is developed to present the simulation model and the results.

5.1 Simulation model

In this thesis we have used event oriented approach for simulation. Event oriented simulation is a useful tool for networking related studies. The followings events were defined for the model

- Packet generation (E1)
- Packet transmission (E2)
- Packet reception (E3)
- Handover packet to Active manager (E4)
- Handover packet to Routing manager (E5)
- En-queue the packet (E6)

Packet generation event occurs at the source node of a traffic flow. The source node generates the packet, and schedules the corresponding packet transmission event. The

packets are generated so as to emulate desired traffic flow. We use a continuous generation of packets at fixed intervals of time.

Packet Transmission event emulates the transmission of a packet on the outgoing interface. This is done by making the outgoing link busy for duration of the transmission interval.

Packet Reception event accepts the incoming packet and sets the status of the incoming link free. The packet is then handed over to diverter. The diverter sends this packet for processing.

Handover Packet to Active Manager event is scheduled by the Packet reception event. The packet is passed to the active manager, which then processes the packet in the context of the session specified by the session id field in the packet header. The active manager then passes this to the routing manager.

Handover to Routing Manager event is scheduled by the active manager. Routing manager selects the particular routing paradigm to be followed for deciding the network interface this packet should go to. In case no particular routing is specified, default routing is used. It then schedules the *En-queue Packet* event.

En-queue Packet event is scheduled by the routing manager to put the packet in the queue corresponding to the outgoing interface. The queue scheduler then schedules the *Packet Transmission* event.

5.2 Topology

In this section, we describe the two network topologies used in this thesis work.

5.2.1 Active Network

Figure 5.1 shows the active network topology we used. All the nodes in the network are active and are capable of performing intelligent functions on the traffic flows. This topology has 17 nodes marked (0 to 16) which are connected by 27 links as shown. There are three sources s1, s2, s3 that generate the traffic flows and three receivers r1, r2, r3 respectively.

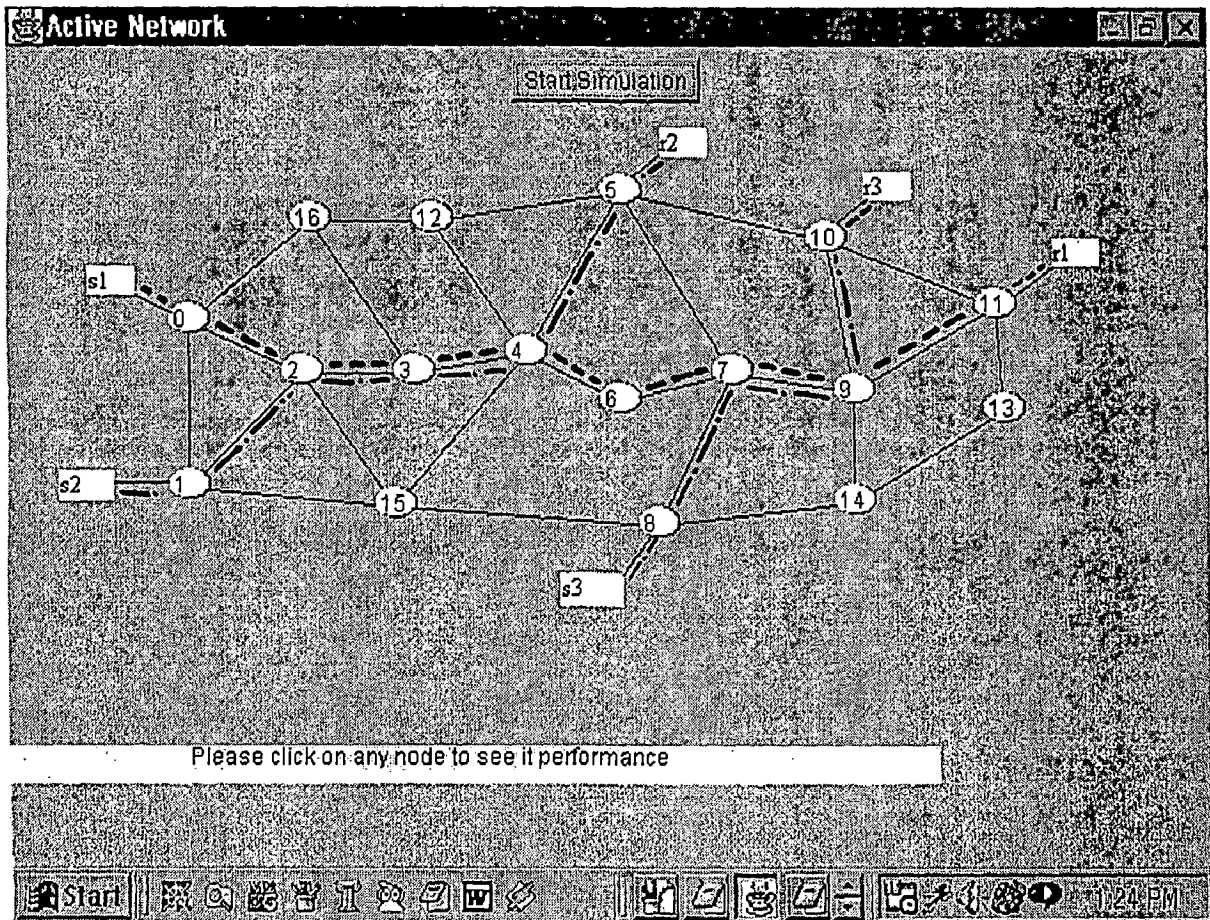


Fig 5.1. Network topology for Active Network

5.2.2 Differentiated services Internet

Figure 5.2 depicts the DS-Internet topology, which is used in this work. It is similar to the Active network topology described earlier except the fact that all the nodes are not active. Only edge nodes 0, 1, 8 are active and are allowed to perform intelligent functions like marking of packets. Same source destination pairs are used to generate traffic, which are used in the active network topology.

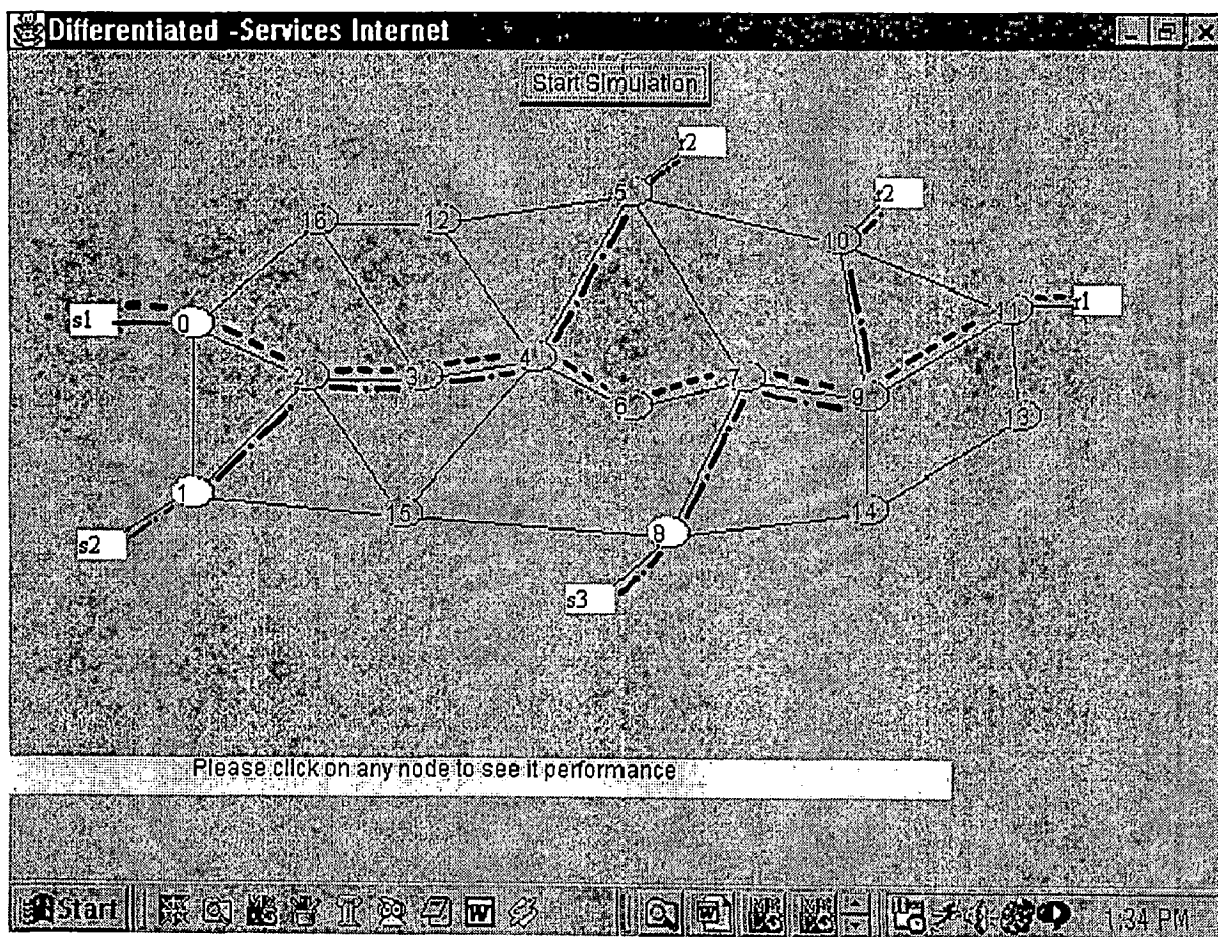


Fig 5.2. Network topology for DS-Internet

The traffic flow from S1 to R1 is the flow with a high priority, which goes through path (0-2-3-4-6-7-9-11). The others, from S2 to R2 flowing through path (1-2-3-4-5) and from S3 to R3 flowing through path (8-7-9-10) are the ones with low priority. We call

them f_0 , f_1 and f_2 respectively. In our scheme, we have taken f_0 as the high priority flow, which has been promised a target throughput and better reliability, while f_1 and f_2 are low priority flows.

In the first part of the simulation, packets are marked in a distributed manner as described in section 4.3. In the second simulation packets are marked at boundary nodes. The experiment simulates congestion at node 7. The relative performance of the two schemes is measured in terms of throughput of each flow.

The graphical user interface is designed using Java's AWT (Abstract Window toolkit)[9] package. This GUI runs under windows environment. The interface is presented as a network topology. The throughput relating to any flow is available to the user, in the form of a graph, by clicking on the particular node. The graph is available during the simulation interval too.

Parameters that we have used for simulation are tabulated in Table 5.1.

Link delay,Bandwidth	2-4 ms,7Mbps
Queue size for each link	30 Packets
Packet size	10KB
Packet processing time	1-3 ms
Simulation time	2500 ms
Target bandwidth for flow f_0	4 Mbps
Target bandwidth for flow f_1	2.75 Mbps
Target bandwidth for flow f_2	3.5 Mbps

Table 5.1 Simulation Parameters

5.3 Implementation details

The software was developed in Java 2[9] under Windows 98 environment. Java is a completely object oriented language which makes simulation very efficient. Further, it is a platform independent language and provides software mobility. Also, JAVA provides built-in packages (Abstract Windowing Toolkit) for a strong GUI support.

In event oriented simulation, an event list is maintained (a link list that contains different types of event) which is sorted by time. An event-scheduler picks up the first event from the event list, executes it and then moves over to the next event. This is done till all the events are exhausted in the list. The link list provides ability to dynamically add and remove events to and from the list.

Each event is an instance of the class event. The Event class contains, an *event type* variable, a reference to the concerned packet, a node id at which the event executes and time of occurrence of the event.

The network model is simulated by a class network, which encapsulates instances of class node and class link. When we start the simulation, packets are generated at different sources, according to type of traffic. In this way, we initiate the event list. The Event Scheduler schedules the first event from the event list. This event may schedule another event, which is inserted in the event list at appropriate place. The simulation time is kept in the form of a long variable. This variable contains time in milliseconds. The Event scheduler increases the value of event clock when it fetches an event from the event list.

Conclusion

6.1 Discussion of results obtained

The simulation results were obtained as graphs depicting throughputs corresponding to different flows measured at different nodes at equal intervals of time.

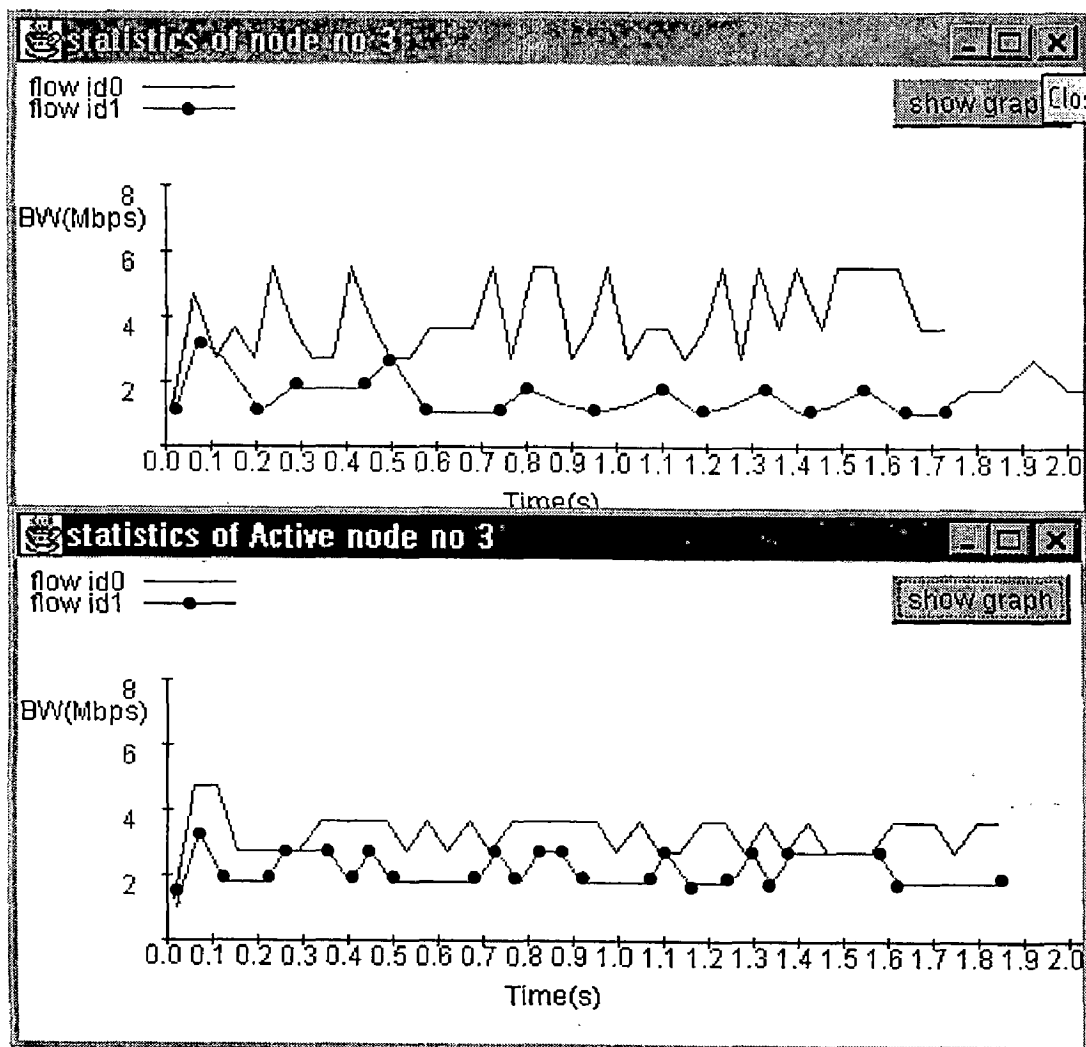


Fig. 6.1 Comparison of throughputs (for f0 and f1) at node 3 between Active network and Internet

The simulated network faces congestion at node 7.

The graphs show statistics measured at node 3 for both the active network and the DS-Internet. It is clear that while the DS-network provides better throughput to the flow f0, it seriously degrades the flow f1. Also there is considerable variation in the throughput. On the other hand, the graph relating to active network shows an improvement in the throughput of the flow f1, with only a minor degradation in the flow f0. Further, the active network implementation also reduces variation in the throughput.

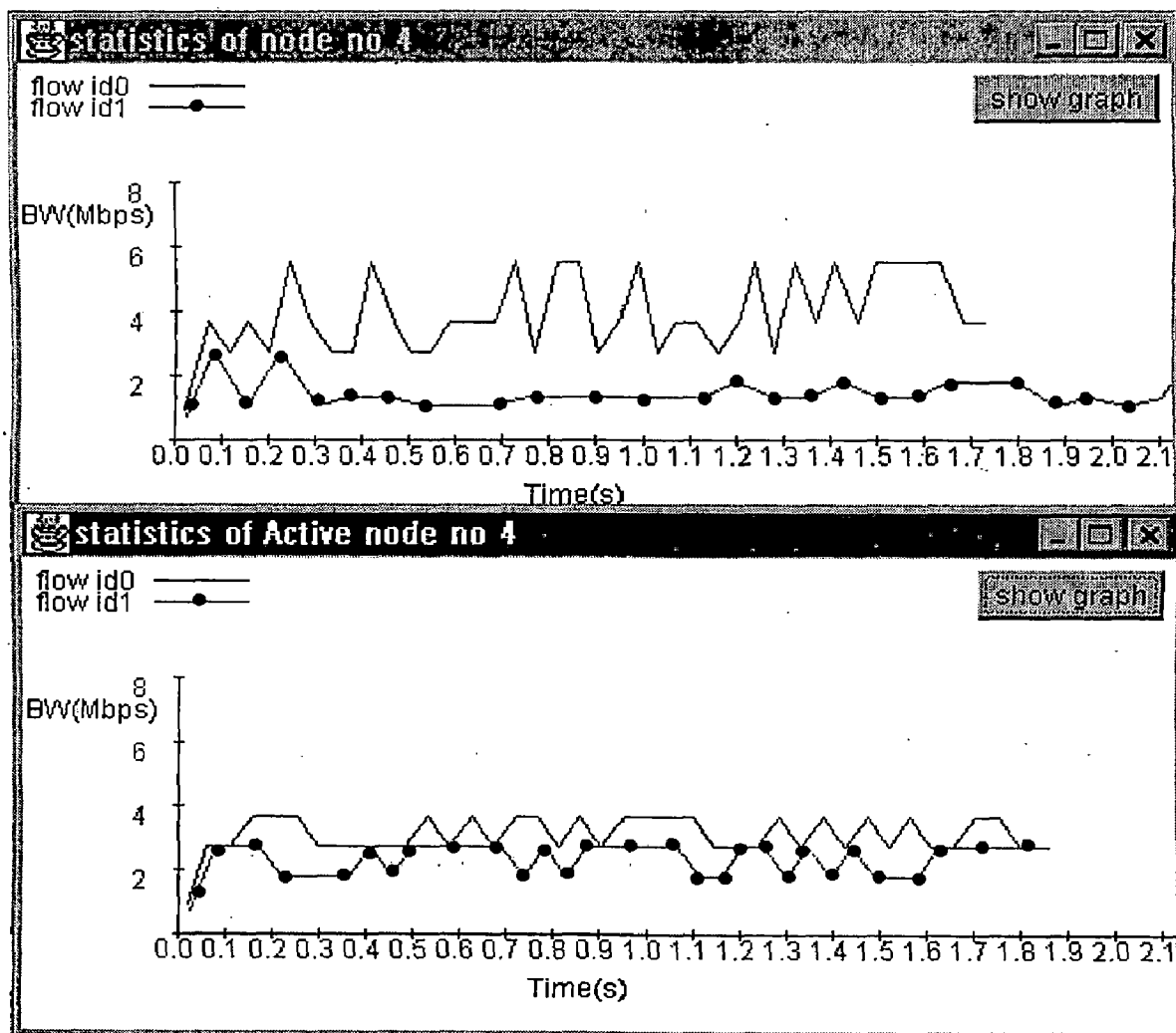


Fig. 6.2 Comparison of throughputs (for f0 and f1) at node 4 between active network and Internet.

Similar results are depicted by the throughput graph, shown in fig 6.2 for node 4.

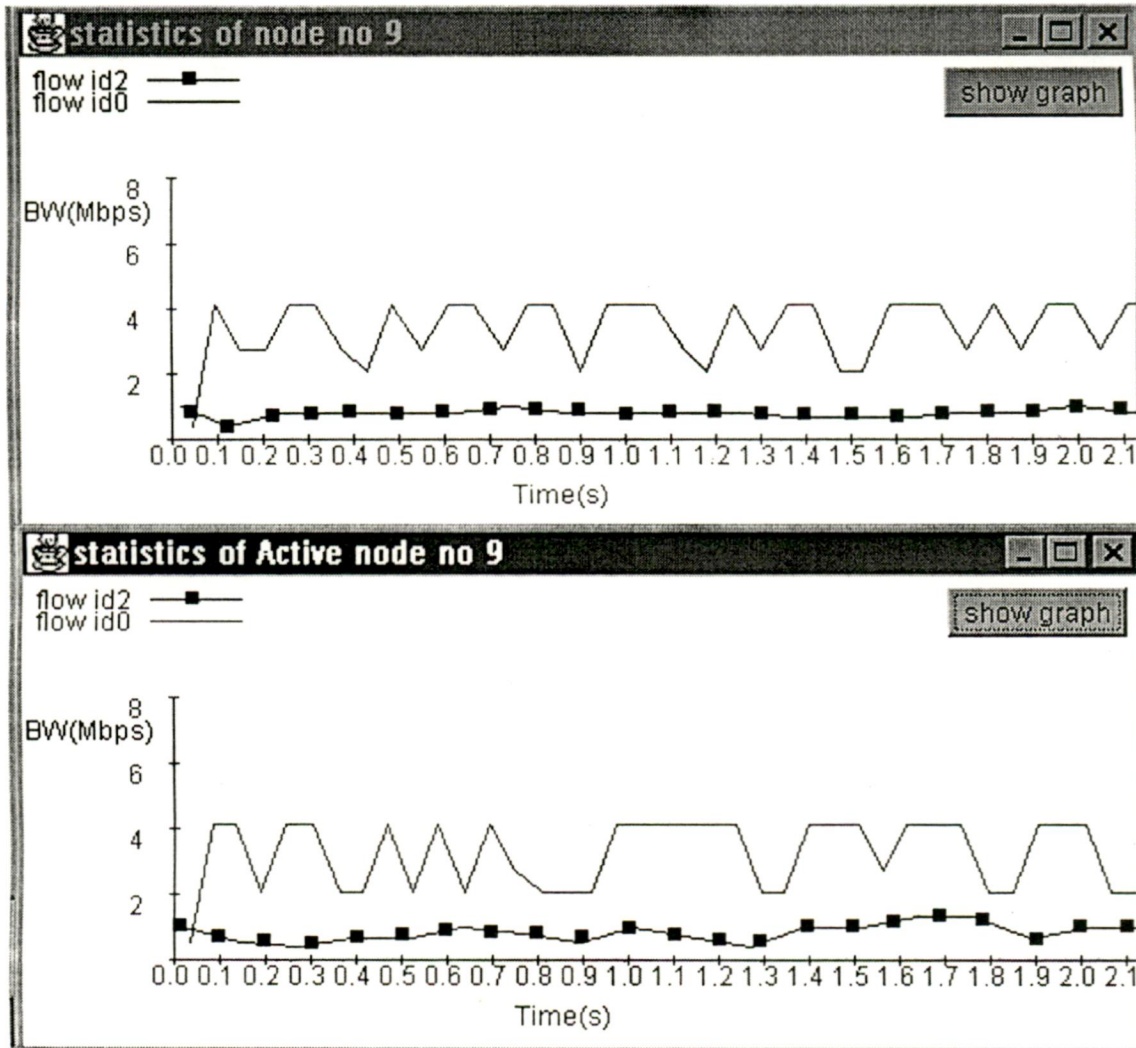


Fig. 6.3 Comparison of throughputs (for f0 and f2) at node 9 between Active network and Internet.

This pair of graphs depicts the throughputs at node 9, which is situated downstream from the congested node 7. The throughput for the flow f2 falls because of congestion, as predictable. Further, the average throughput for the flow f0 is not affected as it is given a preferred treatment here. In both the networks it turns out to be same. This is expected as the node 9 is now receiving marked packets from its upstream neighbor for this flow, in both the networks. The treatment to the marked packets is same in both networks.

Thus the proposed scheme improves network performance by dealing the low priority flows more fairly than the DS-Internet without affecting the treatment given to the high priority flow. So, the overall resource utilization of the network is improved by introducing the concept of active networking.

6.2 Suggestions for future work

In this section we give suggestions for possible extensions in this area.

6.2.1 Multiple code point implementation:

In this thesis we have dealt with one pair of priorities. A more realistic approach will be to use multiple priority code points, in this approach the active node prioritize the packet according to network condition and traffic policies. This may leads to better utilization of resources.

6.2.2 New models and algorithms

The single most prominent disadvantage of active networks is the increase in processing overheads at routers. Thus a lot of work can be done in the design and analysis of new models and algorithms for active networks that can reduce this processing time.

6.2.3 Security issue

The programmable nature of an active network brings legitimate safety and security concern. Incorporating issues related to security can further enhance the implementation of active networks. Incorporating authentication procedures in the routers may be a good future improvement.

Further, new applications for active networking can also be developed.

References

1. D. J. Wetherall and D. L. Tennenhouse, "The **ACTIVE IP Option**", 7th ACM SIGOPS European Workshop, 1996.
2. Yechiam Yemini and Sushil da Silva, "Towards Programmable Networks", IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, October, 1996.
3. D. Scott Alexander, Bob Braden, Carl A. Gunter, Alden W. Jackson, Angelos, D. Keromytis, Gary J. Minden, David Wetherall, "Active Network Encapsulation Protocol (ANEP)", Active Networks Group, Request for Comments: DRAFT.1997
4. D. L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden, "A Survey of Active Network Research", IEEE Communications Magazine, January 1997.
5. U. Legedza, D. Wetherall, and J. Gutttag, "Introducing New Internet services: Why and How", IEEE Network Magazine July/August 1998.
6. D. Black (EMC Corporation), M. Carlson (Sun Microsystems), E. Davies (Nortel UK), S. Blake Torrent Networking Technologies, Z. Wang (Bell Labs Lucent Technologies), W. Weiss (Lucent Technologies), "An Architecture for Differentiated Services", Network Working Group Request for Comments: 2475, December 1998.
7. V.P. Kumar, T. V. Laxaman and D. Stiliadis, "Beyond Best Effort: Router Architecture for differentiated Services of Tomorrow's Internet," IEEE communication mag., vol. 36, no. 5, may 1998.

8. Ulana Legedza, David Wetherall and John Guttag, "Improving the performance of distributed Application Using Active Networks", IEEE INFOCOM, San Francisco, April 1998.
9. Danny Raz and Yuval Shavitt, "Active Network for Efficient Distributed Network Management", IEEE Personal Communication mag. March 2000.
10. Patrick Noughton, Herbert Schildt, "The Complete Reference Java 2", 3rd Edition, TMH, 1999.
11. David Wetherall, John Guttag, David Tennenhouse, "ANTS: Network Services Without the Red Tape", IEEE computer mag., April 1999.
12. D.L. Tennenhouse, S.J. Garland, L. Shrira and M.F. Kaashoek, "From Internet to ActiveNet", Request for Comment, Jan 1996.

G10508



Dsnet.java

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
class Event{
    public int type_of_event;
    public int node_id;
    public int time_of_event;
    public packet pkt;
    public link l;
    Event(int type_of_event,int time_of_event,int node_id,packet pkt){
        this.type_of_event=type_of_event;
        this.node_id=node_id;
        this.time_of_event=time_of_event;
        this.pkt=pkt;
    }
    Event(int type_of_event,int time_of_event,int node_id,link l){
        this.type_of_event=type_of_event;
        this.node_id=node_id;
        this.time_of_event=time_of_event;
        this.l=l;
    }
}

class Event_list{
    private static LinkedList el=new LinkedList();
    public static int clock=0;
    Event_list(){

    }

    public static void initialised_event(int n)
    {
        Random r=new Random(567544);
        int gt=r.nextInt(5);
        packet pkt=new packet(20,1000,0,22,0);
        pkt.tos=1;
        Event event=new Event(2,gt,0,pkt);
        el.add(event);
        for(int i=0;i<n-1;i++){
            pkt=new packet(20,1000,0,22,i+1);
            pkt.tos=1;
            gt=gt+4;
            event=new Event(2,gt,0,pkt);
            put_event(event);
        }
        gt=r.nextInt(5);
        for(int i=0;i<n;i++){
            pkt=new packet(20,1000,1,15,i);
            pkt.tos=0;
            gt=gt+6;
            event=new Event(2,gt,1,pkt);
            put_event(event);
        }
        gt=r.nextInt(5);
        for(int i=0;i<n;i++){
            pkt=new packet(20,1000,2,21,i);
```

```

    pkt.tos=0;
    gt=gt+5;
    event=new Event(2,gt,2,pkt);
    System.out.println("event type"+event.type_of_event+"et="+
        event.time_of_event+"nid"+event.node_id);
    put_event(event);
    }
}

public static void put_event(Event event){
    int i;
    i=find_location(event);
    try{
        el.add(i,event);
    }catch(IndexOutOfBoundsException e){System.out.println("index is not
        allow"+i);}
    }

    private static int find_location(Event event){
    int j=0;

    for(j=0;j<el.size();j++){

        try{
            if(((Event)el.get(j)).time_of_event>event.time_of_event)
                return j;
        }catch(IndexOutOfBoundsException e){System.out.println("index is not allow
            in find location");}
        }

        return j;
    }
}

public static Event get_next_event(){
    Event event;
    event =(Event)el.removeFirst();
    return event;
    }

    public static int getsize(){
    return el.size();
    }
}

class packet{
    public byte version;
    public byte flags;
    public int type_id;
    public int hdr_len;
    public int pkt_len;
    public int source_address;
    public int des_address;
    public long flow_id;
    public byte qos;
    public byte tos;
    public int pkt_no;//for tst
    public int inter_pkt_delay=3;
    packet(){}
```

```

    no_of_unmarked_pkt=0;
    this.targate_throughput= targate_throughput;
}
public void advance_statistics_of_session(packet pkt){
    no_of_pkt++;
    meter.get_to_meter(pkt);
    observed_throughput=meter.get_throughput();
    if(pkt.qos==0)
        no_of_unmarked_pkt++;
    else
        no_of_marked_pkt++;
}
}

class Meter{
    private int throughput;
    public int th_arr[][];
    private int c_time;
    private int l_time;
    private int no_of_pkt;
    private int bit_count;
    private int count;
    Meter(){
        no_of_pkt=0;
        c_time=0;
        l_time=0;
        th_arr=new int[1000][2];
        bit_count=0;
        count=0;
    }
    public void get_to_meter(packet pkt){
        no_of_pkt++;
        bit_count+=pkt.get_pkt_length();
        if(Event_list.clock-l_time>5){
            c_time=Event_list.clock;
            measure_throughput();
        }
    }
    private void measure_throughput(){
        throughput=bit_count/(c_time-l_time);
        l_time=c_time;
        bit_count=0;
        th_arr[count][0]=throughput;
        th_arr[count++][1]=c_time;
    }
    public int get_throughput(){
        return throughput;
    }
    public void print_th_arr(){
        for(int i=0;i<count;i++)
            System.out.print(" "+th_arr[i][0]);System.out.println("");
    }
}

class node{
    private Random r;
    public LinkedList fl;
}

```

```

packet(int hdr_len,int pkt_len,int s_add,int d_add,int pn){
    pkt_no=pn;
    this.hdr_len=hdr_len;
    this.pkt_len=pkt_len;
    source_address=s_add;
    des_address=d_add;
    qos=0;
}
public void assign_flow_id(int flow_id){
    this.flow_id=flow_id;
}
public int get_pkt_length(){
    return pkt_len;
}
public int data(){
    int bw=pkt_len/inter_pkt_delay;
    return bw;
}
}
class source{
    private int sid;
    private int router_id;//through which source is connected to the network
    private int no_of_pkt;
    source(int sid,int rid){
        this.sid=sid;
        no_of_pkt=0;
        router_id=rid;
    }
    public void send(packet pkt){
        no_of_pkt++;
        //System.out.println("send in source");
        pkt.assign_flow_id(sid);
        send_to_router(pkt);
    }
    private void send_to_router(packet pkt){
        Event event=new Event(0,Event_list.clock+1,router_id,pkt);
        Event_list.put_event(event);
    }
}
}
class flow{
    public long flow_id;
    public int s_address;
    public int d_address;
    public int no_of_pkt;
    public int no_of_marked_pkt;
    public int no_of_unmarked_pkt;
    public int targate_throughput;
    public int observed_throughput;
    public Meter meter;
    flow(long flow_id, int s_address,int d_address,int targate_throughput){
        this.flow_id=flow_id;
        this.s_address=s_address;
        this.d_address=d_address;
        meter=new Meter();
        no_of_pkt=0;
        no_of_marked_pkt=0;
    }
}

```

```

public int node_id;
public int no_of_flow;
public int no_of_pkt;
public int no_of_marked_pkt;
public int no_of_unmarked_pkt;
private int route_table[][];
node(int nid){
    r=new Random(34554);
    fl=new LinkedList();
    node_id=nid;
    no_of_flow=0;
    no_of_pkt=0;
    no_of_marked_pkt=0;
    no_of_unmarked_pkt=0;
}
public void receive(packet pkt){
    link l=find_last_hop(pkt);
    l.link_status=false;
    no_of_pkt++;
    if(pkt.qos==0)
        no_of_unmarked_pkt++;
    else
        no_of_marked_pkt++;
    if(node_id==pkt.des_address){
        return;
    }
    send_for_typechacking(pkt);
}
void send_to_activeManager(packet pkt){
    flow f=find_session(pkt);
    f.advance_statistics_of_session(pkt);

    if(pkt.tos>0)
        MarkingEngin.send_for_typechacking(pkt,f);
    int service_time=1+r.nextInt(2); //random generation of service time
    Event event=new Event(4,Event_list.clock+service_time,node_id,pkt);
    Event_list.put_event(event);
}
void send_to_routing_manager(packet pkt){
    link l=find_next_hop(pkt);
    boolean l_status=l.is_busy();
    if(l_status==false&&l.ll.size()==0)
        schedule_send(pkt);

    else{
        int sending_time=l.free_time+(l.delay*l.ll.size());
        if(pkt.qos==1)
            l.ll.addFirst(pkt);
        else
            l.ll.addLast(pkt);
        schedule_send(sending_time,l);
    }
}
void schedule_send(int st,link l){
    Event event=new Event(3,st,node_id,l);
}

```

```

    Event_list.put_event(event);
}
void schedule_send(packet pkt){
    Event event=new Event(1,Event_list.clock,node_id,pkt);
    Event_list.put_event(event);
}

flow find_session(packet pkt){
    flow f;
    try{
        for(int i=0;i<fl.size();i++){
            f=(flow)fl.get(i);

            if(f.flow_id==pkt.flow_id&&f.s_address==pkt.source_address&&f.d_address==p
                kt.des_address)
                return f;
        }
    }catch(IndexOutOfBoundsException e)
        {System.out.println("index is not allow in find location");}
    f=new flow(pkt.flow_id, pkt.source_address,pkt.des_address,pkt.data());
    fl.add(f);
    return f;
}

public int no_of_flow(){
    return fl.size();
}

    public link find_next_hop(packet pkt){
    int nid=get_next_node(pkt);
    link l=Dsnet.Internet.linkarray[0];
    for(int i=1;i<Dsnet.Internet.lc;i++){
        if((l.get_sid()==node_id)&&(l.get_eid()==nid))
            return l;
        l=Dsnet.Internet.linkarray[i];
    }
    return l;
}

public link find_last_hop(packet pkt){
    int l_nid=get_last_node(pkt);
    link l=Dsnet.Internet.linkarray[0];
    for(int i=1;i<Dsnet.Internet.lc;i++){
        if((l.get_sid()==l_nid)&&(l.get_eid()==node_id))
            return l;
        l=Dsnet.Internet.linkarray[i];
    }
    return l;
}

public int send(packet pkt){
    link l=find_next_hop(pkt);
    l.link_status=true;
    l.free_time=Event_list.clock+l.delay;
    Event event=new Event(0,Event_list.clock+l.delay,l.get_eid(),pkt);
    Event_list.put_event(event);
    return l.get_eid();
}

    public int send_from_queue(link l){
    try{
        packet pkt=(packet)l.ll.removeFirst();

```

```

        l.link_status=true;
        l.free_time=Event_list.clock+l.delay;
        Event_event=new Event(0,Event_list.clock+l.delay,l.get_eid(),pkt);
        Event_list.put_event(event);
    }catch(NullPointerException e){System.out.println("error");}
    return l.get_eid();
}

int get_next_node(packet pkt){
    int i=0;
    try{
        for(i=0;i<11;i++){
            if(network.path[(int)pkt.flow_id][i]==node_id)
                break;
        }
    }catch(ArrayIndexOutOfBoundsException e)
        {System.out.println("out of network");}
    return network.path[(int)pkt.flow_id][++i];
}

int get_last_node(packet pkt){
    int i=0;
    try{
        for( i=0;i<11;i++){
            if(network.path[(int)pkt.flow_id][i]==node_id)
                break;
        }
    }catch(ArrayIndexOutOfBoundsException e)
        {System.out.println("out of network");}
    return network.path[(int)pkt.flow_id][--i];
}
}

class MarkingEngin
{
    public static void send_to_marking_engin(packet pkt ,flow f){
        pkt.qos=1;
    }
}

class link{
    private int start_nid;
    public LinkedList ll;
    private int end_nid;
    public int capacity;
    public int delay;
    public boolean link_status;
    public int free_time;//when the link will be free
    link(int st_id,int eid ,int cap,int dlay){
        ll=new LinkedList();
        start_nid=st_id;
        link_status=false;
        free_time=0;
        end_nid=eid;
        capacity=cap;
        delay=dlay;
    }
    int get_sid(){

```

```

        return start_nid;
    }
    int get_eid(){
        return end_nid;
    }
    public boolean is_busy(){
        return link_status;
    }
}

class network{
    Random r;
    public node nodearray[];
    public link linkarray[];
    public source sourcearray[];
    public int lc;//no of link
    public static int
    path[][]={{0,10,12,13,14,16,17,19,21},{1,11,12,13,14,15},{2,18,17,19,20}};
    network(int n){
        r=new Random();
        nodearray=new node[50];
        linkarray=new link[200];
        sourcearray=new source[15];
        for(int i=0;i<n;i++)
            nodearray[i]=new node(i);
        int i=0;
        int ld=3;
        linkarray[lc++]=new link(0,10,10,ld);
        linkarray[lc++]=new link(1,11,10,ld);
        linkarray[lc++]=new link(2,19,10,ld);
        linkarray[lc++]=new link(15,3,10,ld);
        linkarray[lc++]=new link(20,4,10,ld);
        linkarray[lc++]=new link(21,5,10,ld);
        linkarray[lc++]=new link(10,12,10,ld);
        linkarray[lc++]=new link(11,12,10,ld);
        linkarray[lc++]=new link(12,13,10,ld);
        linkarray[lc++]=new link(13,14,10,ld);
        linkarray[lc++]=new link(14,15,10,ld);
        linkarray[lc++]=new link(14,16,10,ld);
        linkarray[lc++]=new link(16,17,10,5);
        linkarray[lc++]=new link(18,17,10,ld);
        linkarray[lc++]=new link(17,19,10,4);
        linkarray[lc++]=new link(19,20,10,ld);
        linkarray[lc++]=new link(19,21,10,ld);
        linkarray[lc++]=new link(20,21,10,ld);
        linkarray[lc++]=new link(23,21,10,ld);
        linkarray[lc++]=new link(19,24,10,ld);
        linkarray[lc++]=new link(24,25,10,ld);
        linkarray[lc++]=new link(10,26,10,ld);
        linkarray[lc++]=new link(10,11,10,ld);
        linkarray[lc++]=new link(26,13,10,ld);
        linkarray[lc++]=new link(26,22,10,ld);
        linkarray[lc++]=new link(22,14,10,ld);
        linkarray[lc++]=new link(22,15,10,ld);
        linkarray[lc++]=new link(15,17,10,ld);
        linkarray[lc++]=new link(15,20,10,ld);
        linkarray[lc++]=new link(12,25,10,ld);
    }
}

```



```

linkarray[lc++]=new link(11,25,10,ld);
linkarray[lc++]=new link(25,14,10,ld);
linkarray[lc++]=new link(25,18,10,ld);
linkarray[lc++]=new link(18,24,10,ld);
linkarray[lc++]=new link(24,19,10,ld);
linkarray[lc++]=new link(24,23,10,ld);

sourcearray[0]=new source(0,10);
sourcearray[1]=new source(1,11);
sourcearray[2]=new source(2,18);
sourcearray[3]=new source(3,10);
sourcearray[4]=new source(4,11);
sourcearray[5]=new source(5,19);
}

}

class gui extends Frame implements
WindowListener, MouseMotionListener,MouseListener{
private node node_arr[];
private link link_arr[];
Button b;
private int gx[]={70,70,130,190,250,300,300,360,320,
425,410,500,200,505,425,180,135};
private int gy[]={138,225,165,165,155,70,180,165,
245,175,95,130,85,185,233,235,85};
private int sx[]={0,0,300,283,530,405};
private int sy[]={105,220,285,0,84,27};
gui(node node_arr[],link link_arr[]){
b=new Button("Start Simulation");
b.addMouseListener(this);
add(b);
addMouseListener(this);
addWindowListener(this);
addMouseMotionListener(this);
setBackground(Color.pink);
setTitle("Differentiated -Services Internet");
this.node_arr=node_arr;
this.link_arr=link_arr;
this.setLayout(new FlowLayout());
setSize(650,420);
setVisible(true);
System.out.print("node_id="+node_arr[2].node_id+
"no_mpkt"+node_arr[2].no_of_marked_pkt);
}

public void mouseClicked(MouseEvent me){}
public void mouseEntered(MouseEvent me){}
public void mouseExited(MouseEvent me){}
public void mousePressed(MouseEvent me){
int mx=me.getX();
int my=me.getY();
for(int i=0;i<17;i++){
if((mx>=gx[i]+20)&&(mx<=gx[i]+40)&&(my>=gy[i]+20)&&(my<=gy[i]+40)){
Graph graph=new Graph(node_arr[i+10],link_arr[i]);link has to find out
break;
}}
}
}

```

```

public void mouseReleased(MouseEvent me){}
public void mouseDragged(MouseEvent me){}
public void mouseMoved(MouseEvent me){}
public void windowActivated(WindowEvent we){}
public void windowClosed(WindowEvent we){}
public void windowDeactivated(WindowEvent we){}
public void windowDeiconified(WindowEvent we){}
public void windowIconified(WindowEvent we){}
public void windowOpened(WindowEvent we){}
public void windowClosing(WindowEvent we){System.exit(0);}
public void paintnet(){
    Graphics g=getGraphics();
    repaint();
    g.setColor(Color.black);
    g.drawString("Please click on any node to see it performance",100,400);

}

    public void paint(Graphics g){
try{
g.setColor(Color.black);

for(int i=3,sx,sy,ex,ey;i<32;i++){

    sx=gx[link_arr[i].get_sid()-10]+20;
    sy=gy[link_arr[i].get_sid()-10]+20;
    ex=gx[link_arr[i].get_eid()-10]+20;
    ey=gy[link_arr[i].get_eid()-10]+20;
    g.setColor(Color.black);
    g.drawLine(sx+10,sy+10,ex+10,ey+10);

}catch(Exception e){System.out.println("lskd");}
try{
for(int i=0;i<17;i++){
g.setColor(Color.blue);
g.drawOval(gx[i]+20,gy[i]+20,21,15);
g.setColor(Color.white);
g.fillOval(gx[i]+20,gy[i]+20,20,15);
}
g.setColor(Color.blue);
for(int i=0;i<17;i++)
    g.drawString(new Integer(i).toString(),gx[i]+22,gy[i]+33);
g.setColor(new Color(255,255,200));
g.fillRect(0,390,500,20);
catch(Exception e){System.out.println("skd");}

g.setColor(Color.black);
g.drawString("Please click on any node to see it performance",100,400);
}

public void movepkt(int startnode,int nextnode){
    int i;Graphics g1=getGraphics();

    int x1=gx[startnode-10];
    int x2=gx[nextnode-10];
    int y1=gy[startnode-10];

```

```

int y2=gy[nextnode-10];
int x3=x1;
int y3=y1;
if(x2==x1){
    if(y2>y1)
        {
            for(i=(int)y1;i<(int)y2;i+=3){
                try{

                    g1.setColor(Color.blue);
                    g1.drawRect((int)x3-7,(int)y3,6,10);
                    Thread.sleep(100);
                    g1.setColor(getBackground());
                    g1.drawRect((int)x3-7,(int)y3,6,10);

                    y3=y3+5;

                }catch(InterruptedException e){}
            }
        }
    if(y2<y1)
        {
            for(i=(int)y2;i<(int)y1;i+=3){
                try{

                    g1.setColor(Color.blue);
                    g1.drawRect((int)x3-7,(int)y3,6,10);
                    Thread.sleep(100);
                    g1.setColor(getBackground());
                    g1.drawRect((int)x3-7,(int)y3,6,10);

                    y3=y3-5;
                }catch(InterruptedException e){}
            }
        }
}
if(x2>x1){int m=0;

    for(i=20;i<(x2-x1)-6;){
        try{x3=x1+i;
            if(y2!=y1)
                {
                    m=6;
                    i+=1;
                }
            else{
                i+=3;
                m=0;
            }
        }
        g1.setColor(Color.blue);
        g1.drawRect((int)x3+m,(int)y3-1,10,6);
        Thread.sleep(0);
        g1.setColor(getBackground());
        g1.drawRect((int)x3+m,(int)y3-1,10,6);
        y3=((y2-y1)/(x2-x1))*i+y1;
        if(y2!=y1)

```



```

public void plot_graph(){
    Graphics g=getGraphics();
    int ox=70;
    int oy=200;
    int scaled=4;
    int x,y;
    for(int j=0;j<n.fl.size();j++)
    {
        flow f=(flow)n.fl.get(j);
        g.setColor(Color.black);
        g.drawString("flow id"+f.flow_id,10,40+j*10);
        if(f.flow_id==0)
            g.setColor(Color.blue);
        else
            if(f.flow_id==1)
                g.setColor(Color.magenta);
            else
                g.setColor(Color.black);
        g.drawLine(60,35+j*10,100,35+j*10);
    }
    g.setColor(Color.red);
    g.drawLine(ox,oy,ox,80);
    g.drawLine(ox,oy,600,oy);
    for(int i=ox,j=0;i<=600;i+=20,j+=100){
        g.drawLine(i,oy-2,i,oy+2);
        g.drawString(""+(float)j/1000,i-10,oy+12);
    }
    for(int i=80,j=4;i<=oy;i+=30,j--){
        g.drawLine(ox-2,i,ox+2,i);
        if(j!=0)
            g.drawString(""+j*2,ox-20,i+10);
    }
    g.setColor(Color.black);
    g.drawString("Time(s)",ox+150,oy+30);
    g.drawString("BW(Mbps)",ox-65,oy-100);
    g.setColor(Color.blue);
    for(int j=0;j<n.fl.size();j++)
    {
        for(int i=0;i<195&&(((flow)n.fl.get(j)).meter.th_arr[i][1]<10000);i=i+5){
            if(((flow)n.fl.get(j)).flow_id==0)
            {
                g.setColor(Color.blue);
            }
            else
                if(((flow)n.fl.get(j)).flow_id==1)
                {
                    g.setColor(Color.magenta);
                }
            else
                g.setColor(Color.black);
            g.drawLine(ox+(((flow)n.fl.get(j)).meter.th_arr[i][1]/4),oy-
                (((flow)n.fl.get(j)).meter.th_arr[i][0]/4),ox+(((flow)n.fl.get(j)).m
                eter.th_arr[i+5][1]/4),oy-low(n.fl.get(j)).meter.th_arr[i+5][0]/4));
        }
    }
}
public void mouseEntered(MouseEvent me){}

```

```

public void mouseExited(MouseEvent me){}
public void mousePressed(MouseEvent me){
    Object source=me.getSource();
    if(source==b){
        plot_graph();
    }
}
public void mouseReleased(MouseEvent me){}
public void mouseDragged(MouseEvent me){}
public void mouseMoved(MouseEvent me){}
public void windowActivated(WindowEvent we){}
public void windowClosed(WindowEvent we){}
public void windowDeactivated(WindowEvent we){}
public void windowDeiconified(WindowEvent we){}
public void windowIconified(WindowEvent we){}
public void windowOpened(WindowEvent we){}
public void windowClosing(WindowEvent we){
    setVisible(false);
}
}

```

```

class Dsnet {
    public static network Internet;

    public static void main(String arg[])throws Exception{
        Internet=new network(50);
        gui animation=new gui(Internet.nodearray,Internet.linkarray);
        Event_list.initialised_event(500);
        int d=0;
        while(Event_list.getsize()!=0){//simulation start
            try{
                Event event=Event_list.get_next_event();
                Event_list.clock=event.time_of_event;
                if(event.type_of_event==0){
                    Internet.nodearray[event.node_id].receive(event.pkt);
                }
                if(event.type_of_event==1)
                {
                    d=Internet.nodearray[event.node_id].send(event.pkt);

                    animation.movepkt(Internet.nodearray[event.node_id].node_id,d);

                }
                if(event.type_of_event==2)
                    Internet.sourcearray[event.node_id]. send(event.pkt);
                if(event.type_of_event==3)
                {
                    Internet.nodearray[event.node_id].send_from_queue(event.l);
                }
                if(event.type_of_event==4) ,

                Internet.nodearray[event.node_id].send_to_routing_manager(event.pkt);

            }catch(IndexOutOfBoundsException e){System.out.println("error");}
            animation.paintnet();
        }
    }
}

```

```
for(int i=10;i<16;i++){
    node n=Internet.nodearray[i];
    system.out.print("@@node_id="+Internet.nodearray[i].node_id+"no_mpkt
    "+Internet.nodearray[i].no_of_marked_pkt+"!!"+i+"!!!!"+Internet.node
    array[3].no_of_marked_pkt);
for(int j=0;j<n.fl.size();j++){
    flow f=(flow)n.fl.get(j);
    System.out.print("node_id="+Internet.nodearray[i].node_id);
    System.out.print("flow_id"+f.flow_id+"mar pkt"+f.no_of_marked_pkt);
    f.meter.print_th_arr();
    }
    }//end of while
} //end of main

} // end of class
```

Simulate.java

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
class Event
{
    public int type_of_event;
    public int node_id;
    public int time_of_event;
    public smart_pkt spkt;
    public link l;
    Event(int type_of_event,int time_of_event,int node_id,smart_pkt spkt){
        this.type_of_event=type_of_event;
        this.node_id=node_id;
        this.time_of_event=time_of_event;
        this.spkt=spkt;
    }
    Event(int type_of_event,int time_of_event,int node_id,link l)
    {
        this.type_of_event=type_of_event;
        this.node_id=node_id;
        this.time_of_event=time_of_event;
        this.l=l;
    }
}

class Event_list
{
    private static LinkedList el=new LinkedList();
    //public static int event_no=0;
    public static int clock=0;
    Event_list(){}

    public static void initialised_event(int n)
    {
        Random r=new Random(567544);
        int gt=r.nextInt(5);
        smart_pkt spkt=new smart_pkt(20,1000,0,22,0);
        spkt.tos=1;
        Event event=new Event(2,gt,0,spkt);
        System.out.println("event type"+event.type_of_event+"et="
        +event.time_of_event+"nid"+event.node_id);
        el.add(event);
        for(int i=0;i<n-1;i++){
            spkt=new smart_pkt(20,1000,0,22,i+1);
            spkt.tos=1;
            gt=gt+4;
            event=new Event(2,gt,0,spkt);
```



```

//System.out.println(""+i);
put_event(event);
    }
    gt=r.nextInt(5);
    for(int i=0;i<n;i++)
    {
    spkt=new smart_pkt(20,1000,1,15,i);
    spkt.tos=0;
    gt=gt+6;
    event=new Event(2,gt,1,spkt);
    //System.out.println("event type"+event.type_of_event+"et="+
    event.time_of_event+"nid"+event.node_id);
    put_event(event);
    }
    gt=r.nextInt(5);
    for(int i=0;i<n;i++){
    spkt=new smart_pkt(20,1000,2,21,i);
    spkt.tos=0;
    gt=gt+5;
    event=new Event(2,gt,2,spkt);
    System.out.println("event type"+event.type_of_event+
    "et="+event.time_of_event+"nid"+event.node_id);
    put_event(event);
    }
}

```

```

public static void put_event(Event event){
    int i;
    i=find_location(event);
    try{
        el.add(i,event);
        System.out.println("in pe"+i+" "+el.size());
    }catch(IndexOutOfBoundsException e)
        {System.out.println("index is not allow"+i);}
}

```

```

private static int find_location(Event event)
{
    int j=0;
    for(j=0;j<el.size();j++)
    {
        System.out.println("in fl"+j+" "+el.size());
        try{
            if(((Event)el.get(j)).time_of_event>event.time_of_event)
                return j;
        }catch(IndexOutOfBoundsException e)
            {System.out.println("index is not allow in find location");}
    }
}

return j;

```

```

    }

    public static Event get_next_event(){
        Event event;
        event =(Event)el.removeFirst();
        return event;
    }

    public static int getsize(){
        return el.size();
    }
}

class smart_pkt{
    public byte version;
    public byte flags;
    public int type_id;
    public int hdr_len;
    public int pkt_len;
    public int source_address;
    public int des_address;
    public long flow_id;
    public byte qos;
    public byte tos;
    public int pkt_no;//for tst
    public int inter_pkt_delay=3;
    smart_pkt(){
    smart_pkt(int hdr_len,int pkt_len,int s_add,int d_add,int pn){
        pkt_no=pn;
        this.hdr_len=hdr_len;
        this.pkt_len=pkt_len;
        source_address=s_add;
        des_address=d_add;
        qos=0;
    }

    public void assign_flow_id(int flow_id){
        this.flow_id=flow_id;
    }
    public int get_pkt_length(){
        return pkt_len;
    }
    public int data(){
        int bw=pkt_len/inter_pkt_delay;
        return bw;
    }
}

class source{
    private int sid;
    private int router_id;//through which source is connected to the network
    private int no_of_pkt;

```

```

source(int sid,int rid){
    this.sid=sid;
    no_of_pkt=0;
    router_id=rid;
}
public void send(smart_pkt spkt){
    no_of_pkt++;
    //System.out.println("send in source");
    spkt.assign_flow_id(sid);
    send_to_router(spkt);
}
private void send_to_router(smart_pkt spkt){
    Event event=new Event(0,Event_list.clock+1,router_id,spkt);
    Event_list.put_event(event);
}
}

class flow{
    public long flow_id;
    public int s_address;
    public int d_address;
    public int no_of_pkt;
    public int no_of_marked_pkt;
    public int no_of_unmarked_pkt;
    public int targate_throughput;
    public int observed_throughput;
    public Meter meter;
    flow(long flow_id, int s_address,int d_address,int targate_throughput){
        this.flow_id=flow_id;
        this.s_address=s_address;
        this.d_address=d_address;
        meter=new Meter();
        no_of_pkt=0;
        no_of_marked_pkt=0;
        no_of_unmarked_pkt=0;
        this.targate_throughput= targate_throughput;
    }
    public void advance_statistics_of_session(smart_pkt spkt)
    {
        no_of_pkt++;
        meter.get_to_meter(spkt);
        if(no_of_pkt%5==0){
            observed_throughput=meter.get_throughput();
        }
        if(spkt.qos==0)
            no_of_unmarked_pkt++;
        else
            no_of_marked_pkt++;
    }
}

```

```

class Meter{
    private int throughput;
    public int th_arr[][];
    private int c_time;
    private int l_time;
    private int no_of_pkt;
    private int bit_count;
    private int count;

    Meter()
    {
        no_of_pkt=0;
        c_time=0;
        l_time=0;
        th_arr=new int[1000][2];
        bit_count=0;
        count=0;
    }
    public void get_to_meter(smart_pkt spkt)
    {
        no_of_pkt++;
        bit_count+=spkt.get_pkt_length();
        if(Event_list.clock-l_time>5){
            if(no_of_pkt%5==0){
                c_time=Event_list.clock;
                measure_throughput();
            }
        }
    }
    private void measure_throughput()
    {
        throughput=bit_count/(c_time-l_time);
        l_time=c_time;
        bit_count=0;
        th_arr[count][0]=throughput;
        th_arr[count++][1]=c_time;
        System.out.println("count"+count+" "+throughput);
    }
    public int get_throughput()
    {
        return throughput;
    }
}

```

```

public void print_th_arr(){
    for(int i=0;i<count;i++)
        System.out.print(" "+th_arr[i][0]);System.out.println("");
    }
}

```

```

class node{
    private Random r;
    public LinkedList fl;
    public int node_id;
    public int no_of_flow;
    public int no_of_pkt;
    public int no_of_marked_pkt;
    public int no_of_unmarked_pkt;
    private int route_table[][];
    public boolean next_hop_congestion;
    private boolean is_congestion;
    node(int nid)
    {
        r=new Random(34554);
        fl=new LinkedList();
        node_id=nid;
        no_of_flow=0;
        no_of_pkt=0;
        no_of_marked_pkt=0;
        no_of_unmarked_pkt=0;
        next_hop_congestion=false;
    }
    public int[][] get_routetable()
    {
        return route_table;
    }
    public void receive(smart_pkt spkt)
    {
        send_to_diverter(smart_pkt spkt);
        link l=find_last_hop(spkt);
        l.link_status=false;
        no_of_pkt++;
        if(spkt.qos==0)
            no_of_unmarked_pkt++;
        else
            no_of_marked_pkt++;
        if(node_id==spkt.des_address){
            //System.out.println("pkt has received");
            return;}
        send_to_activeManager(spkt);
        //System.out.println("receive in node"+node_id+" "+no_of_pkt);
    }
    void send_to_diverter(smart_pkt spkt){
        if(spkt.type_id== 0){
            send_to_routing_manager(spakt);}
    }
}

```

```

    if(spkt.tos>0) send_to_marking_engin(spkt,f);
    MarkingEngin.send_to_marking_engin(spkt,f);
    int service_time=1+r.nextInt(2);//random generation of service time
    Event event=new Event(4,Event_list.clock+service_time,node_id,spkt);
    Event_list.put_event(event);
    }
public void send_to_marking_engin(smart_pkt spkt ,flow f){

    if(is_congesion )
    {
    spkt.qos=1;
    else
    spkt.qos=0;
    }
void send_to_routing_manager(smart_pkt spkt){
    link l=find_next_hop(spkt);
    boolean l_status=l.is_busy();
    if(l_status==false&&l.ll.size()==0)
        schedule_send(spkt);

    else
    {
        int sending_time=l.free_time+(l.delay*l.ll.size());
        System.out.println(""+l.ll.size());
        if(spkt.qos==1)
            l.ll.addFirst(spkt);
        else
            l.ll.addLast(spkt);
        schedule_send(sending_time,l);
    }
}
void schedule_send(int st,link l){
    Event event=new Event(3,st,node_id,l);
    Event_list.put_event(event);
}
void schedule_send(smart_pkt spkt){
    Event event=new Event(1,Event_list.clock,node_id,spkt);
    Event_list.put_event(event);
}

flow find_session(smart_pkt spkt){
    flow f;
    try{
    for(int i=0;i<fl.size();i++){
    System.out.println("queue size"+fl.size());
    f=(flow)fl.get(i);

    if(f.flow_id==spkt.flow_id&&f.s_address==spkt.source_address&&f.d_address==spkt.des_address)

```

```

        return f;
    }
} catch(IndexOutOfBoundsException e)
    {System.out.println("index is not allow in find location");}
f=new flow(spkt.flow_id, spkt.source_address,spkt.des_address,spkt.data());
fl.add(f);
return f;
}
public int no_of_flow()
{
    return fl.size();
}
public link find_next_hop(smart_pkt spkt)
{
    int nid=get_next_node(spkt);
    link l=simulate.activenet.linkarray[0];
    for(int i=1;i<simulate.activenet.lc;i++){
        if((l.get_sid()==node_id)&&(l.get_eid()==nid))
            return l;
        l=simulate.activenet.linkarray[i];
    }
    return l;
}
public link find_last_hop(smart_pkt spkt){
    int l_nid=get_last_node(spkt);
    link l=simulate.activenet.linkarray[0];
    for(int i=1;i<simulate.activenet.lc;i++){
        if((l.get_sid()==l_nid)&&(l.get_eid()==node_id))
            return l;
        l=simulate.activenet.linkarray[i];
    }
    return l;
}
public int send(smart_pkt spkt)
{
    //System.out.println("send to node"+node_id);
    //int nid=get_next_node(spkt);
    link l=find_next_hop(spkt);
    l.link_status=true;
    l.free_time=Event_list.clock+l.delay;
    Event event=new Event(0,Event_list.clock+l.delay,l.get_eid(),spkt);
    Event_list.put_event(event);
    return l.get_eid();
}
public int send_from_queue(link l)
{
    //System.out.println("send to node"+node_id);
    //int nid=get_next_node(spkt);
    //link l=find_next_hop(n_id);
    try{
        smart_pkt spkt=(smart_pkt)l.ll.removeFirst();
    }
}

```

```

l.link_status=true;
l.free_time=Event_list.clock+l.delay;
Event event=new Event(0,Event_list.clock+l.delay,l.get_eid(),spkt);
Event_list.put_event(event);
}catch(NullPointerException e){System.out.println("8888888deepak i a m goyal");}
return l.get_eid();
}
int get_next_node(smart_pkt spkt)
{
int i=0;
try{
for( i=0;i<11;i++){//System.out.println("&&&&&" +network.path[(int)spkt.flow_id][i]);
if(network.path[(int)spkt.flow_id][i]==node_id)
break;
}
}catch(ArrayIndexOutOfBoundsException e){System.out.println("out of network");}
return network.path[(int)spkt.flow_id][++i];
}
int get_last_node(smart_pkt spkt){
int i=0;
try{
for( i=0;i<11;i++){//System.out.println("&&&&&" +network.path[(int)spkt.flow_id][i]);
if(network.path[(int)spkt.flow_id][i]==node_id)
break;
}
}catch(ArrayIndexOutOfBoundsException e){System.out.println("out of network");}
return network.path[(int)spkt.flow_id][--i];
}
}

```

```

class MarkingEngin{
public static void send_to_marking_engin(smart_pkt spkt ,flow f){
spkt.qos=1;
if(next_hop_congestion==true){

if(f.observed_throughput<f.targate_throughput){
spkt.qos=1;
}
else
spkt.qos=0;
}
}
}

```

```

class link{
private int start_nid;
public LinkedList ll;
private int end_nid;
public int capacity;
public int delay;
public boolean link_status;
}

```



```

public int free_time;//when the link will be free
link(int st_id,int eid ,int cap,int dlay){
ll=new LinkedList();
start_nid=st_id;
link_status=false;
free_time=0;
end_nid=eid;
capacity=cap;
delay=dlay;
}
int get_sid()
{
return start_nid;
}
int get_eid()
{
return end_nid;
}
public boolean is_busy()
{
return link_status;
}
}

```

```

class network

```

```

{
Random r;
public node nodearray[];
public link linkarray[];
public source sourcearray[];
public int lc;//no of link
public static int path[][]={{0,10,12,13,14,16,17,19,21},{1,11,12,13,14,15},{2,18,17,19,20}};
network(int n){
r=new Random();
nodearray=new node[50];
linkarray=new link[200];
sourcearray=new source[15];
for(int i=0;i<n;i++)
nodearray[i]=new node(i);
int i=0;
int ld=3;
linkarray[lc++]=new link(0,10,10,ld);
linkarray[lc++]=new link(1,11,10,ld);
linkarray[lc++]=new link(2,19,10,ld);
/*
linkarray[lc++]=new link(15,3,10,ld);
linkarray[lc++]=new link(20,4,10,ld);
linkarray[lc++]=new link(21,5,10,ld);*/
linkarray[lc++]=new link(10,12,10,ld);
linkarray[lc++]=new link(11,12,10,ld);
linkarray[lc++]=new link(12,13,10,ld);
linkarray[lc++]=new link(13,14,10,ld);
}
}

```

```

linkarray[lc++]=new link(14,15,10,ld);
linkarray[lc++]=new link(14,16,10,ld);
linkarray[lc++]=new link(16,17,10,ld);
linkarray[lc++]=new link(18,17,10,ld);
linkarray[lc++]=new link(17,19,10,4);
linkarray[lc++]=new link(19,20,10,ld);
linkarray[lc++]=new link(19,21,10,ld);
linkarray[lc++]=new link(20,21,10,ld);
linkarray[lc++]=new link(23,21,10,ld);
linkarray[lc++]=new link(19,24,10,ld);
//linkarray[lc++]=new link(24,25,10,ld);
linkarray[lc++]=new link(10,26,10,ld);
linkarray[lc++]=new link(10,11,10,ld);
linkarray[lc++]=new link(26,13,10,ld);
linkarray[lc++]=new link(26,22,10,ld);
linkarray[lc++]=new link(22,14,10,ld);
linkarray[lc++]=new link(22,15,10,ld);
linkarray[lc++]=new link(15,17,10,ld);
linkarray[lc++]=new link(15,20,10,ld);
linkarray[lc++]=new link(12,25,10,ld);
linkarray[lc++]=new link(11,25,10,ld);
linkarray[lc++]=new link(25,14,10,ld);
linkarray[lc++]=new link(25,18,10,ld);
linkarray[lc++]=new link(18,24,10,ld);
linkarray[lc++]=new link(24,19,10,ld);
linkarray[lc++]=new link(24,23,10,ld);

```

```

sourcearray[0]=new source(0,10);
sourcearray[1]=new source(1,11);
sourcearray[2]=new source(2,18);
sourcearray[3]=new source(3,10);
sourcearray[4]=new source(4,11);
sourcearray[5]=new source(5,19);

```

```

}

```

```

}

```

```

class gui extends Frame implements

```

```

    WindowListener, MouseMotionListener,MouseListener{

```

```

    private node node_arr[];

```

```

    private link link_arr[];

```

```

    Button b;

```

```

    private int gx[]={70,70,130,190,250,300,300,360,320,425,410,500,200,505,425,180,135};

```

```

    private int gy[]={138,225,165,165,155,70,180,165,245,175,95,130,85,185,233,235,85};

```

```

    private int sx[]={0,0,300,283,530,405};

```

```

    private int sy[]={105,220,285,0,84,27};

```

```

gui(node node_arr[],link link_arr[]){

```

```

    b=new Button("Start Simulation");

```

```

    b.addMouseListener(this);

```

```

    add(b);

```

```

    addMouseListener(this);

```

```

    addWindowListener(this);

```

```

addMouseMotionListener(this);
setBackground(Color.pink);
setTitle("Active Network");

this.node_arr=node_arr;
this.link_arr=link_arr;
this.setLayout(new FlowLayout());
setSize(650,420);
setVisible(true);
System.out.print("node_id="+node_arr[2].node_id+"no_mpkt"+node_arr[2].no_of_marked_pkt);

}
public void mouseClicked(MouseEvent me){}
public void mouseEntered(MouseEvent me){}
public void mouseExited(MouseEvent me){}
public void mousePressed(MouseEvent me){
int mx=me.getX();
int my=me.getY();
for(int i=0;i<17;i++){
if((mx>=gx[i]+20)&&(mx<=gx[i]+40)&&(my>=gy[i]+20)&&(my<=gy[i]+40)){
Graph graph=new Graph(node_arr[i+10],link_arr[i]);//link has to find out
break;
}}
}
public void mouseReleased(MouseEvent me){}
public void mouseDragged(MouseEvent me){}
public void mouseMoved(MouseEvent me){}
public void windowActivated(WindowEvent we){}
public void windowClosed(WindowEvent we){}
public void windowDeactivated(WindowEvent we){}
public void windowDeiconified(WindowEvent we){}
public void windowIconified(WindowEvent we){}
public void windowOpened(WindowEvent we){}
public void windowClosing(WindowEvent we){System.exit(0);}
public void paintnet(){
Graphics g=getGraphics();
repaint();
g.setColor(Color.black);
g.drawString("Please click on any node to see it performance",100,400);

}
public void paint(Graphics g){
try{
g.setColor(Color.black);

for(int i=0,sx1,sy1,ex1,ey1;i<6;i++){
sx1=sx[link_arr[i].get_sid()-10]+20;
sy1=sy[link_arr[i].get_sid()-10]+20;
ex1=sx[link_arr[i].get_eid()-10]+20;
ey1=sy[link_arr[i].get_eid()-10]+20;

```

```

        g.setColor(Color.black);
        g.drawLine(sx1+10,sy1+10,ex1+10,ey1+10);
    }
    for(int i=3,sx,sy,ex,ey;i<32;i++){

        sx=gx[link_arr[i].get_sid()-10]+20;
        sy=gy[link_arr[i].get_sid()-10]+20;
        ex=gx[link_arr[i].get_eid()-10]+20;
        ey=gy[link_arr[i].get_eid()-10]+20;
        g.setColor(Color.black);
        g.drawLine(sx+10,sy+10,ex+10,ey+10);
    }
} catch(Exception e){System.out.println("lskd");}
    try{
        for(int i=0;i<6;i++){
            g.setColor(Color.black);
            g.drawRect(sx[i],sy[i],20,20);
        }
        for(int i=0;i<17;i++){
            g.setColor(Color.blue);
            g.drawOval(gx[i]+20,gy[i]+20,21,15);
            g.setColor(Color.white);
            g.fillOval(gx[i]+20,gy[i]+20,20,15);
        }
        g.setColor(Color.blue);
        for(int i=0;i<17;i++){
            g.drawString(new Integer(i).toString(),gx[i]+22,gy[i]+33);
            g.setColor(new Color(255,255,200));
            g.fillRect(0,390,500,20);
        }
} catch(Exception e){System.out.println("skd");}

g.setColor(Color.black);
g.drawString("Please click on any node to see it performance",100,400);
}

public void movepkt(int startnode,int nextnode){
    int i;Graphics g1=getGraphics();

    int x1=gx[startnode-10];
    int x2=gx[nextnode-10];
    int y1=gy[startnode-10];
    int y2=gy[nextnode-10];
    System.out.println("***** "+gx[startnode-10]+" "+gx[nextnode-10]+" "+gy[startnode-10]+"
"+gy[nextnode-10]);
    int x3=x1;
    int y3=y1;
    if(x2==x1){
        if(y2>y1)
            {

```

```

addMouseListener(this);
setBackground(Color.pink);
setTitle("Active Network");

this.node_arr=node_arr;
this.link_arr=link_arr;
this.setLayout(new FlowLayout());
setSize(650,420);
setVisible(true);
System.out.print("node_id="+node_arr[2].node_id+"no_mpkt"+node_arr[2].no_of_marked_pkt);

}
public void mouseClicked(MouseEvent me){}
public void mouseEntered(MouseEvent me){}
public void mouseExited(MouseEvent me){}
public void mousePressed(MouseEvent me){
int mx=me.getX();
int my=me.getY();
for(int i=0;i<17;i++){
if((mx>=gx[i]+20)&&(mx<=gx[i]+40)&&(my>=gy[i]+20)&&(my<=gy[i]+40)){
Graph graph=new Graph(node_arr[i+10],link_arr[i]);//link has to find out
break;
}}
}
public void mouseReleased(MouseEvent me){}
public void mouseDragged(MouseEvent me){}
public void mouseMoved(MouseEvent me){}
public void windowActivated(WindowEvent we){}
public void windowClosed(WindowEvent we){}
public void windowDeactivated(WindowEvent we){}
public void windowDeiconified(WindowEvent we){}
public void windowIconified(WindowEvent we){}
public void windowOpened(WindowEvent we){}
public void windowClosing(WindowEvent we){System.exit(0);}
public void paintnet(){
Graphics g=getGraphics();
repaint();
g.setColor(Color.black);
g.drawString("Please click on any node to see it performance",100,400);

}
public void paint(Graphics g){
try{
g.setColor(Color.black);

for(int i=0,sx1,sy1,ex1,ey1;i<6;i++){
sx1=sx[link_arr[i].get_sid()-10]+20;
sy1=sy[link_arr[i].get_sid()-10]+20;
ex1=sx[link_arr[i].get_eid()-10]+20;
ey1=sy[link_arr[i].get_eid()-10]+20;

```

```

        g.setColor(Color.black);
        g.drawLine(sx1+10,sy1+10,ex1+10,ey1+10);
    }
    for(int i=3,sx,sy,ex,ey;i<32;i++){

        sx=gx[link_arr[i].get_sid()-10]+20;
        sy=gy[link_arr[i].get_sid()-10]+20;
        ex=gx[link_arr[i].get_eid()-10]+20;
        ey=gy[link_arr[i].get_eid()-10]+20;
        g.setColor(Color.black);
        g.drawLine(sx+10,sy+10,ex+10,ey+10);
    }catch(Exception e){System.out.println("lskd");}
    try{
        for(int i=0;i<6;i++){
            g.setColor(Color.black);
            g.drawRect(sx[i],sy[i],20,20);
        }
        for(int i=0;i<17;i++){
            g.setColor(Color.blue);
            g.drawOval(gx[i]+20,gy[i]+20,21,15);
            g.setColor(Color.white);
            g.fillOval(gx[i]+20,gy[i]+20,20,15);
        }
        g.setColor(Color.blue);
        for(int i=0;i<17;i++){
            g.drawString(new Integer(i).toString(),gx[i]+22,gy[i]+33);
            g.setColor(new Color(255,255,200));
            g.fillRect(0,390,500,20);
        }catch(Exception e){System.out.println("skd");}

    g.setColor(Color.black);
    g.drawString("Please click on any node to see it performance",100,400);
    }

    public void movepkt(int startnode,int nextnode){
        int i;Graphics g1=getGraphics();

        int x1=gx[startnode-10];
        int x2=gx[nextnode-10];
        int y1=gy[startnode-10];
        int y2=gy[nextnode-10];
        System.out.println("***** "+gx[startnode-10]+" "+gx[nextnode-10]+" "+gy[startnode-10]+"
        "+gy[nextnode-10]);
        int x3=x1;
        int y3=y1;
        if(x2==x1){
            if(y2>y1)
                {

```

```

for(i=(int)y1;i<(int)y2;i+=3){
    try{

        g1.setColor(Color.blue);
        g1.drawRect((int)x3-7,(int)y3,6,10);
        Thread.sleep(100);
        g1.setColor(getBackground());
        g1.drawRect((int)x3-7,(int)y3,6,10);

        y3=y3+5;//10->5

    }catch(InterruptedException e){}
}

}

if(y2<y1)
{
    for(i=(int)y2;i<(int)y1;i+=3){
        try{

            g1.setColor(Color.blue);
            g1.drawRect((int)x3-7,(int)y3,6,10);
            Thread.sleep(100);
            g1.setColor(getBackground());
            g1.drawRect((int)x3-7,(int)y3,6,10);

            y3=y3-5;//10->5

        }catch(InterruptedException e){}
    }
}

if(x2>x1){int m=0;//workable

    for(i=20;i<(x2-x1)-6;){
        try{x3=x1+i;
            if(y2!=y1)
            {
                m=6;
                i+=1;
            }
            else{
                i+=3;
                m=0;
            }
            g1.setColor(Color.blue);
            g1.drawRect((int)x3+m,(int)y3-1,10,6);
            Thread.sleep(0);
            g1.setColor(getBackground());

```

```

        g1.drawRect((int)x3+m,(int)y3-1,10,6);
        y3=((y2-y1)/(x2-x1))*i+y1;
        if(y2!=y1)
        {
            m=6;
            i+=1;
        }
        else{
            i+=3;
            m=0;
        }
    }catch(InterruptedException e){System.out.println("***99999*");}

}

}

if(x2<x1){
    for(i=6;i<(x1-x2);i+=3){
        try{

            g1.setColor(Color.blue);
            g1.drawRect((int)x3,(int)y3-5,6,10);
            Thread.sleep(1);
            g1.setColor(getBackground());
            g1.drawRect((int)x3,(int)y3-5,6,10);
            x3=x1-i;
            y3=((y2-y1)/(x2-x1))*(-i)+y1;

        }catch(InterruptedException e){}
    }
}

}

}

class Graph extends Frame implements
    WindowListener, MouseMotionListener,MouseListener{
    int x;
    int y;
    node n;
    link l;
    Button b;

    Graph(node n,link l){

        Font f=new Font("Times New Roman",Font.BOLD,8);
        setFont(f);
        setBackground(Color.gray);
        b=new Button("show graph");
    }
}

```



```

g.drawString("Time(s)",ox+150,oy+30);
g.drawString("BW(Mbps)",ox-65,oy-100);
g.setColor(Color.blue);System.out.println("hi "+n.no_of_pkt+" "+n.fl.size());
for(int j=0;j<n.fl.size();j++)
{
    for(int i=0;i<195&&(((flow)n.fl.get(j)).meter.th_arr[i][1]<10000);i=i+5)
    {
        System.out.println("hi ");
        System.out.println(" "+ox+" "+(oy-(((flow)n.fl.get(j)).meter.th_arr[i][0]/4)));
    }
    if(((flow)n.fl.get(j)).flow_id==0)
    {
        g.setColor(Color.black);
        g.fillOval(ox+i,oy-(((flow)n.fl.get(j)).meter.th_arr[i][0]/4),5,5);
        g.setColor(Color.blue);
        g.drawLine(ox+(((flow)n.fl.get(j)).meter.th_arr[i][1]/4),oy-
(((flow)n.fl.get(j)).meter.th_arr[i][0]/4),ox+(((flow)n.fl.get(j)).meter.th_arr[i+5][1]/4),oy-
(((flow)n.fl.get(j)).meter.th_arr[i+5][0]/4));
        g.drawLine(ox+i,oy-(((flow)n.fl.get(j)).meter.th_arr[i][0]/4),ox+i+5,oy-
(((flow)n.fl.get(j)).meter.th_arr[i+5][0]/4));
    }
    else
    if(((flow)n.fl.get(j)).flow_id==1)
    {
        g.setColor(Color.magenta);
    }
    else
    g.setColor(Color.black);
    g.fillRect(ox+i,oy-(((flow)n.fl.get(j)).meter.th_arr[i][0]/4),5,5);
    g.drawLine(ox+i,oy-(((flow)n.fl.get(j)).meter.th_arr[i][0]/4),ox+i+5,oy-
(((flow)n.fl.get(j)).meter.th_arr[i+5][0]/4));
    g.drawLine(ox+(((flow)n.fl.get(j)).meter.th_arr[i][1]/4),oy-
(((flow)n.fl.get(j)).meter.th_arr[i][0]/4),ox+(((flow)n.fl.get(j)).meter.th_arr[i+5][1]/4),oy-
(((flow)n.fl.get(j)).meter.th_arr[i+5][0]/4));
}
}
}

public void mouseEntered(MouseEvent me){}
public void mouseExited(MouseEvent me){}
public void mousePressed(MouseEvent me){
    Object source=me.getSource();
    if(source==b){
        plot_graph();
    }
}

public void mouseReleased(MouseEvent me){}
public void mouseDragged(MouseEvent me){}
public void mouseMoved(MouseEvent me){}
public void windowActivated(WindowEvent we){}
public void windowClosed(WindowEvent we){}
public void windowDeactivated(WindowEvent we){}

```

```

b.addMouseListener(this);
add(b);
addMouseListener(this);
addWindowListener(this);
addMouseMotionListener(this);
this.n=n;
this.l=l;
this.setLayout(new FlowLayout(FlowLayout.RIGHT));
setSize(500,250);
setTitle("statistics of Active node no "+((n.node_id)-10));
setVisible(true);
}
public void paint(Graphics g1){}
public void mouseClicked(MouseEvent me){}

public void plot_graph(){
Graphics g=getGraphics();

setFont(new Font("Times New Roman",Font.BOLD,10));
int ox=70;
int oy=200;//180
int scaled=4;
int x,y;
for(int j=0;j<n.fl.size();j++)
{
flow f=(flow)n.fl.get(j);
g.setColor(Color.black);
g.drawString("flow id"+f.flow_id,10,40+j*10);
if(f.flow_id==0)
g.setColor(Color.blue);
else
if(f.flow_id==1)
g.setColor(Color.magenta);
else
g.setColor(Color.black);
g.drawLine(60,35+j*10,100,35+j*10);
g.drawString("no of marked pkt="+f.no_of_marked_pkt,10,40+j*10);
}
g.setColor(Color.red);
g.drawLine(ox,oy,ox,80);
g.drawLine(ox,oy,600,oy);
for(int i=ox,j=0;i<=600;i+=20,j+=100){
g.drawLine(i,oy-2,i,oy+2);
g.drawString(""+(float)j/1000,i-10,oy+12);
}
for(int i=80,j=4;i<=oy;i+=30,j--){
g.drawLine(ox-2,i,ox+2,i);
if(j!=0)
g.drawString(""+j*2,ox-20,i+10);
}
g.setColor(Color.black);

```

```

public void windowDeiconified(WindowEvent we){}
public void windowIconified(WindowEvent we){}
    public void windowOpened(WindowEvent we){}
    public void windowClosing(WindowEvent we){
        setVisible(false);
    }
}

class simulate {
    public static network activenet;
    public static void main(String arg[])throws Exception{
        activenet=new network(50);
        gui animation=new gui(activenet.nodearray,activenet.linkarray);
        try{Thread.sleep(1000);
        Event_list.initialised_event(500);
        for(int i=0;i<Event_list.getsize();i++){
        Event event=Event_list.get_next_event(i);
            System.out.println("event type"+event.type_of_event+" etime="+event.time_of_event+"
nid"+event.node_id);
        }
        int d=0;
            while(Event_list.getsize()!=0){//simulation start
            try{
            Event event=Event_list.get_next_event();
            Event_list.clock=event.time_of_event;
            if(event.type_of_event==0){
                activenet.nodearray[event.node_id].receive(event.spkt);
            }
            if(event.type_of_event==1)
            {
                d=activenet.nodearray[event.node_id].send(event.spkt);
                // animation.movepkt(activenet.nodearray[event.node_id].node_id,d);
                //activenet.nodearray[event.node_id].send(event.spkt));
            }
            if(event.type_of_event==2)
                activenet.sourcearray[event.node_id]. send(event.spkt);
            if(event.type_of_event==3)
            {
                // animation.movepkt(activenet.nodearray[event.node_id].node_id,
                //activenet.nodearray[event.node_id].send_from_queue(event.l));

                activenet.nodearray[event.node_id].send_from_queue(event.l);
            }
            if(event.type_of_event==4)
                activenet.nodearray[event.node_id].send_to_routing_manager(event.spkt);

            }catch(IndexOutOfBoundsException e){System.out.println("i deepak goyal");}
        }animation.paintnet();

        System.out.println(" after "+Event_list.getsize() );
    }
}

```

```

for(int i=0;i<Event_list.getsize();i++){

Event event=Event_list.get_next_event();
if(event.node_id==14&&event.type_of_event==0){
Thread.sleep(50);

System.out.print("event type"+event.type_of_event);//
System.out.print(" etime="+event.time_of_event+" nid"+event.node_id);
try{
System.out.println("pkt no"+event.spkt.pkt_no);
} catch(NullPointerException e){System.out.println("deepak goyal"+event.type_of_event);}
}}

```

```

for(int i=0;i<Event_list.getsize();i++)
System.out.println(""+Event_list.get_next_event(i)+" after "+Event_list.getsize() );
System.out.println(""+Event_list.getsize());
for(int i=0;i<Event_list.getsize();i++){
Event event=Event_list.get_next_event(i);
System.out.print(" "+event.node_id+"&"+event.time_of_event+"&"+event.type_of_event);

```

```

} catch(NullPointerException e){System.out.println("deepak goyal");}
for(int i=network.path[0][0],j=1;j<9;i=network.path[0][j++){}
node n=activenet.nodearray[i];
flow f=n.fl.get(0);
f.meter.print_th_arr();
}
for(int i=network.path[1][0],j=1;j<5;i=network.path[1][j++){}
node n=activenet.nodearray[i];
flow f=n.fl.get(1);
f.meter.print_th_arr();
}
for(int i=10;i<16;i++){
node n=activenet.nodearray[i];
System.out.print("node_id="+activenet.nodearray[i].node_id+"no_mpkt"
+activenet.nodearray[i].no_of_marked_pkt);
for(int j=0;j<n.fl.size();j++){
flow f=(flow)n.fl.get(j);
System.out.print("node_id="+activenet.nodearray[i].node_id);
System.out.print("flow_id"+f.flow_id+"mar pkt"+f.no_of_marked_pkt);
f.meter.print_th_arr();
}
}

```

```

gui disco=new gui(activenet.nodearray,activenet.linkarray);
}
}

```