Resource Management in Differentiated Services

A Prototype Implementation

Martin Jacobsson

Master of Science Thesis
University of Twente
Computer Science/TSS

June 28, 2001

Examination Committee:
Prof. Dr. Ir. Ignas Niemegeers (UT)
Dr. Ir. Geert Heijenk (UT/Ericsson)
Ir. Hamad el Allali (UT)
Ir. Georgios Karagiannis (Ericsson)

ABSTRACT

This thesis describes the development and implementation of a resource management scheme in a Differentiated Services (Diffserv) domain. This scheme is called Resource Management in Diffserv (RMD) and can be used for dynamic edge-to-edge resource provisioning in Diffserv domains.

Differentiated Services is a Quality of Service (QoS) management architecture that offers particular guarantees for services in an aggregated manner rather than per flow. It achieves scalability by pushing as much of the complexity as possible to the edges of the network domain. Service differentiation is achieved by using a field in the IP header and by defining per-hop behaviors (PHB) for each value. Packets are handled at each node according to the PHB indicated by the field in the message header.

When dynamic quantitative resource reservations are needed, the Diffserv domain needs to be extended with resource management. One such scheme is the Load Control protocol proposed in IETF during the year 2000. RMD, its successor, enhances the Load Control scheme to a complete solution. The RMD scheme consists of two protocols, a per-domain reservation (PDR) protocol and per-hop reservation (PHR) protocol.

- A per-domain reservation protocol is used to perform resource reservations in the complete Diffserv domain. A PDR protocol is used only by the edge nodes and ignored by all interior nodes.
- A per-hop reservation protocol is used to perform per-hop reservations and extends the Diffserv PHB. A PHR protocol is used and implemented in all nodes in the Diffserv domain, including both edge and interior nodes.

Currently, only one PHR protocol is defined in RMD, the RMD On-demand Reservation (RODA) protocol. The resources are requested dynamically per traffic class and reserved on demand in all RMD nodes in the communication path.

RMD is an open and flexible resource management scheme that can be used in many types of Diffserv-enabled IP networks. In particular, RMD can be used to provide resource management in an IP-based UMTS Radio Access Network (UTRAN). The UTRAN network is a wired portion of the UMTS cellular network, and its boundaries are a collection of radio base stations and the gateways to the core transport network. Today, UTRAN uses a circuit-switched transport technology, but in the future, an IP-based transport network might be preferable. In that case, some kind of resource management scheme needs to be used, so that the network can offer good quality to the real-time traffic. This thesis explains why the current IP-based dynamic resource management schemes, such as Integrated Services, cannot be used and why RMD is needed.

The main goal of the assignment was to first specify the RMD architecture in detail and then implement an RMD prototype using the RODA PHR. Furthermore, a simple QoS resource reservation protocol was designed. This protocol, called Simple External Protocol (SEP), was used on an end-to-end basis. Moreover, a PDR protocol that is able to inter-operate with the RODA PHR and the SEP protocol was specified and implemented.

The implementation was made in Linux by extending the current traffic control framework in the kernel. Some basic functional experiments were performed on the prototype to demonstrate the basic RMD functionality.

PREFACE

This thesis is the result of a nine month long final thesis project for the Department of Computer Science at University of Twente, The Netherlands. The project was conducted at the Wireless Multimedia Research Lab in Ericsson EuroLab Netherlands, Enschede, The Netherlands. The project is a part of my M.Sc. program in computer science at Linköpings universitet, Sweden and was made possible by the Erasmus student exchange program.

For this final thesis work, I would like to thank my examination committee for valuable inputs and comments. They are: Prof. Dr. Ir. Ignas Niemegeers, Dr. Ir. Geert Heijenk, Ir. Hamad el Allali and Ir. Georgios Karagiannis.

I am very grateful to my supervisor at Ericsson, Georgios Karagiannis, for the countless hours that he dedicated to this thesis during the entire time. It greatly improved the content of this thesis. I would like to thank him for long interesting conversations and discussions, his ideas, comments and critics, his support with my thesis and for arranging nice pub tours.

And of course, all my colleagues at Ericsson deserve a great thanks. Especially the colleagues in the RMD project group: Vlora Rexhepi and Simon Oosthoek. I would like to thank them and the rest of the group, who are scattered all over the world, for making RMD happen. Without them and their work, RMD would not be much. I also thank Neill Whillans for taking the time to at least partially check my English.

My stay in The Netherlands would have been very boring without friends. I met a lot of new friends from countries all over the world, including The Netherlands, and want to thank you for all interesting conversations and fun time. Most of you will go back to your home countries and I just want to say that I already miss you. Vishy deserves some extra thanks, since he is the best Indian Ph.D. student I ever met.

Then, I would like to give my parents a huge hug. I promise to visit you soon, but before that, I want to thank you for your care, understanding and because you believe in me. Of course my only brother needs a thank. He is quite a special brother after all.

Finally, I want to thank you, Wieke for your very special care and love.

CONTENTS

LIST OF ABBREVIATIONS

3GPP	Third Generation Partnership Project
AAL	ATM Adaptation Layer
AF	Assured Forwarding
ATM	Asynchronous Transfer Mode
CBQ	Class Based Queuing
COPS	Common Open Policy Service
DPS	Dynamic Packet States
DRP	Dynamic Reservation Protocol
DSCP	Differentiated Services Codepoint
EF	Expedited Forwarding
FC-DS	Feedback Controlled Differentiated Services
FIFO	First In First Out
GRED	Generalized Random Early Drop
GSM	Global System for Mobile Communication
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IP	Internet Protocol
MBAC	Measurement-based Admission Control
MPEG	Motion Picture Expert Group
NAT	Network Address Translator
PDB	Per Domain Behavior
PDR	Per Domain Reservation
PHB	Per Hop Behavior
PHR	Per Hop Reservation
QoS	Quality of Service
RAN	Radio Access Network
RAT	Robust Audio Tool
RED	Random Early Drop
RFC	Request for Comment
RMD	Resource Management in Diffserv
RNC	Radio Network Controller
RODA	RMD on Demand
RSVP	Resource Reservation Protocol
RTP	Real-time Transport Protocol
SDH	Synchronous Digital Hierarchy
SEP	Simple External Protocol
SLA	Service Level Agreement
SMS	Short Message Service
SNMP	Simple Network Management Protocol

SRP	Scalable Resource Reservation Protocol
TBF	Token Bucket Filter
TCP	Transmission Control Protocol
TOS	Type of Service
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunication System
UTRAN	UMTS Terrestrial Radio Access Network
WRR	Weighted Round Robin
YESSIR	Yet Another Sender Session Internet Reservations

[1] INTRODUCTION

This thesis describes the development and implementation of a new dynamic resource reservation mechanism, called Resource Management in Diffserv (RMD) [WeJa01a]. It is a dynamic resource management scheme for an edge-to-edge Differentiated Services (Diffserv) domain [RFC2475]. By using resource reservation, different quality of services (QoS) requirements can be satisfied for traffic with high real-time demands. Typical requirements are low delay and small packet losses. RMD can be used in many Internet Protocol (IP) [RFC791] network environments, but it is optimized for use in IP-based cellular networks in particular.

A cellular network is the network that provides wireless communication to mobile users who are moving among different radio coverage areas. These areas are called cells and are controlled by a radio base station. Examples of typical cellular network technologies are Global System for Mobile Communication (GSM) [MoPa92] and Universal Mobile Telecommunication System (UMTS) [3GPP-25401].

Currently, the transmission medium between radio base stations and other equipment in the cellular network is circuit-based. Popular techniques are Synchronous Digital Hierarchy (SDH) and Asynchronous Transfer Mode (ATM). But the rapidly growing popularity of IP and its flexibility to meet future demands makes it a good candidate to be used in the cellular networks of tomorrow.

If an operator uses an IP-based transport network (see also [PaKa01]) between nodes in a cellular network, it gives him the opportunity to upgrade the cellular network with packet-based services. Compared with a traditional circuit-based system, the gain is seen in the statistical aggregation of traffic that can be achieved in a packet-based network and the efficiency for new services. This will result in increased transmission efficiency, reduced leasing costs for the mobile operator and a bigger variety of services for the users.

There are, however, very high requirements on the cellular network. The network has to transfer radio-frames and other quality sensitive data between different cellular specific nodes. A very high percentage of the traffic is delay sensitive and sensitive for packet drops, so the transport services provided by an IP-based network must be as good as the services provided by a circuit–based network. To satisfy this requirement, the IP-based network must use a simple and scalable resource management scheme for the support of real-time traffic. Moreover, it should be ensured that there are adequate transport resources on the links available in the network to handle the real-time traffic requirements. In the situation that the network has limited bandwidth and does not use any mechanism for managing the network resources, congestion may occur which will degrade the network performance.

1.1 Resource Reservation

Quality of service can be defined in different ways and the most commonly used definition is the following. QoS is the collective effect of service performances, which determine the

degree of satisfaction of a user of the service. One way of offering QoS is by using resource reservation. Resource reservation is used to satisfy QoS requirements by dynamically reserving resources in the network nodes in a way that resources are not over-utilized. If the network becomes over-utilized, the end-to-end network performance, such as delay and packet loss, for each flow is severely degraded. This is avoided by reservations. When a resource is reserved by one user, no other user can use that resource before the reservation is released. If some resources are unreserved, it means that they are available and can be reserved by another user for transmitting data.

Resource reservations are usually related to flows. A flow is a session of data traffic from a sending end-host, called sender, to a receiving end-host, called receiver. A flow identifier is used to uniquely identify a flow. It can be any subset of the following parameters: source address, destination address, protocol, source port number and destination port number. In combination with the flow identifier, a specification of the flow traffic characteristics is needed. Examples of such parameters are bandwidth, allowed delay, maximum packet size, etc.

1.2 The Cellular Radio Access Network

The cellular network and especially the cellular Radio Access Network (RAN) has come into major focus because of the increasing demand for bandwidth from the mobile users and the new kind of services they demand. The RAN supports the communication access between a mobile terminal and the core network. This transmission network is already the single highest cost for a mobile operator, which also makes all kinds of improvements interesting.

The RAN depicted in Figure 1-1 is the UMTS Terrestrial Radio Access Network (UTRAN) [3GPP-25401] which is used in UMTS. There is also a RAN available for GSM, which looks similar. However, for simplicity reasons and for not confusing the reader with different types of terminology, only the radio access network for UMTS will be used here [3GPP-21905].

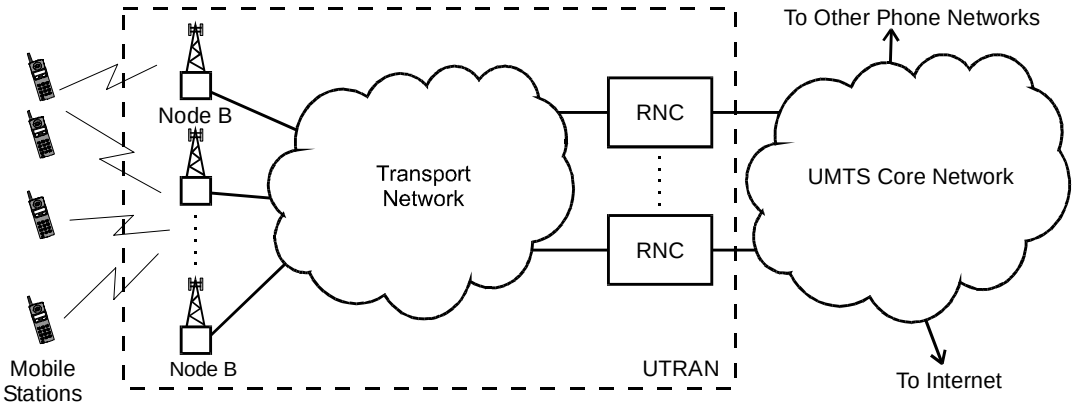

Figure 1-1: Cellular radio access network

The Node B is a radio base station that includes the radio interface functionality for communication with the mobile stations. It consists of an antenna, some hardware and is connected to a radio network controller (RNC) via the transport network. The Node B samples the radio traffic and sends it to the RNC as radio frames. At the same time it receives radio frames from the RNC for transmission over the radio interface to the mobile stations. A radio frame is a short data segment that must be delivered in a timely fashion with limited

delay over the transport network. Otherwise, the frames are discarded by the radio base station or the RNC. The traffic is therefore very sensitive to delays.

An RNC controls a number of radio base stations, managing their traffic on the radio interface, including handovers in the radio access network. Handover is the process applied when a mobile station moves between different cells. This also means that the wireless communication is changed amongst different radio base stations. When such a change occurs, the new radio base station has to establish a new connection to the RNC in the transport network. The old radio base station will, at the same time, release its connection. In UMTS, it is possible for the mobile stations to transmit radio signals to several radio base stations at the same time. All these radio base stations transfer the same radio signal as frames to the RNC and the RNC combines all the frames to obtain a better signal than if only one signal from one radio base station was used.

The UTRAN has to connect all the radio base stations, which means that it must span an entire nation or a big province. If at the same time, good radio coverage is required, it means that several thousands of radio base stations are needed. Therefore, the UTRAN might be very big, both in terms of distances and in terms of number of nodes. The cellular operator usually decides which kind of network topology he will use, based on radio coverage and cost matters. Therefore, a diversity of topologies in the UTRANs are possible. Currently, only ATM is specified for use in the UTRAN but IP has also been discussed. In the following sections, both proposals are described.

1.2.1 ATM-based UTRAN

The current transport medium in UTRAN, specified by the UMTS release 99, is ATM. Figure 1-2 shows how the protocol layers can look like in UTRAN according to release 99. Note that the UMTS release 99 already specifies UDP/IP [RFC791], [RFC768] as the transmission protocol in the core network, but not inside the UTRAN.

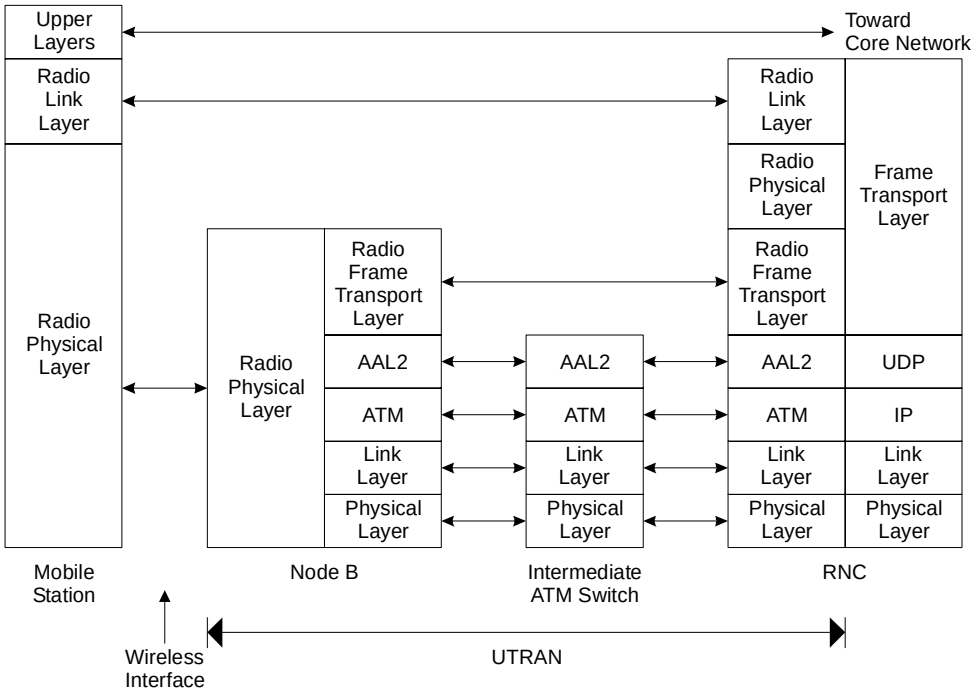

Figure 1-2: Data plane protocol layers in ATM-based UTRAN

The radio physical layer performs radio transmission and reception in the air interface between the mobile station and the radio base station. The radio base station samples the radio signals into radio frames. The radio frame transport layer is used to transmit these frames between the Node B and the RNC. This transportation is done over ATM, using ATM adaptation layer 2 (AAL2) in UMTS release 99. The radio link layer includes extra radio functionality that only the mobile station and the RNC uses. It includes packet segmentation and re-assembly, plus some radio resource control.

The upper layers are the data traffic that the mobile station wants to transfer. It can be anything from voice or SMS, to other data such as web traffic. If the mobile station is IP-aware, the upper layers will contain an IP-layer for the end-to-end data traffic. That IP-layer has nothing to do with the UTRAN transmission and will not be seen by the Node B or the RNC since it is hidden inside the radio frames. That IP-layer will only be seen by the mobile station and the gateway between the UMTS core network and the Internet.

1.2.2 IP-based UTRAN

IP is an open transport architecture that can run on top of many various link layer techniques. The operator can choose among Ethernet, SDH, ATM and much more. The choice is completely transparent to the higher layers, making it possible to use several solutions at the same time. If IP is used everywhere in the UMTS network, then one single network management system can be used. The simple network management protocol (SNMP), defined by IETF [RFC1157], [RFC2570], is very popular in the Internet and can be used to manage both the core network and the radio access networks and its equipment.

One more major advantage of IP is the transmission efficiency. In the UTRAN environment, the traffic paths between the RNC and several Node Bs can share bandwidth. Statistical multiplexing used in the network will result in a reduction of the overall required bandwidth. Even more transmission efficiency with IP can be obtained by using header compression. With recent technologies, it is also possible to offer different kinds of QoS guarantees and different traffic treatment.

Due to the benefits of using an IP-based transport medium instead of ATM, 3GPP has reconsidered the UTRAN architecture. Therefore, UMTS release 2000 will also specify a version of UTRAN that uses an IP-based transport medium. However, to do that, some shortcomings of IP must be solved, which will be explained in section 1.3. Figure 1-3 shows how the protocol layers may look like in such an environment. The changes are marked with gray.

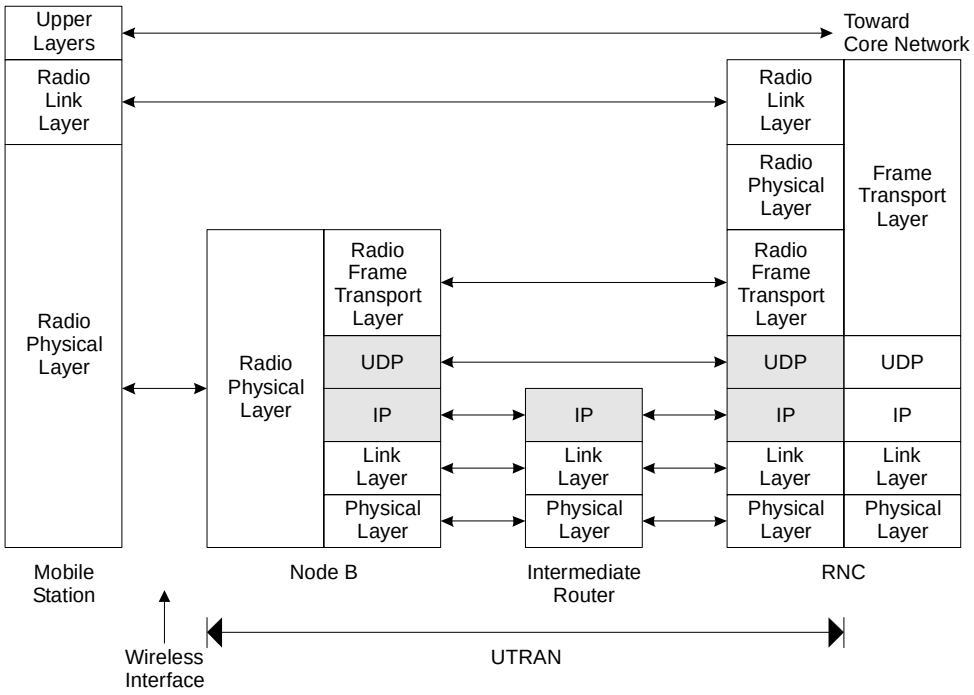

Figure 1-3: Data plane protocol layers in IP-based UTRAN

The only difference is the transport between the Node B and the RNC. ATM and AAL2 have been substituted with UDP/IP and all the radio layers remains the same. In this case, the circuit-based ATM network has been exchanged into a packet-based IP network. This means that the RAN will have some extra requirements on the IP network, which are not offered by usual IP networks.

1.3 RAN Requirements on an IP-based Network

The specific characteristics of UTRAN, such as small delays, few packet losses and high utilization, add extra requirements to an IP network. Basically, the same behavior must be provided by IP as the behavior provided by the circuit-based network. Below, a list is given of minimum requirements for an IP-based UTRAN. These requirements are described in detail in [PaKa01].

- On demand and dynamic allocation of resources. For real-time services, it must be certain that a portion of network resources is available to them. These resources can be reserved on a static or dynamic basis, or potentially based on some kind of measurement of network load. By using static allocations, you must use over-provisioning techniques that will not give a high utilization. The traffic flows are continuously shifting because of the mobility of mobile stations and handovers, and therefore a dynamic resource management scheme is required.
- Good QoS provisioning. The resource management scheme must be able to keep the real-time traffic under a certain pre-defined network utilization in order to avoid congestion.
- Efficient network utilization. Since the cost of the leased line transmission is high, the network must be utilized to the highest possible degree and this must be facilitated by the resource management scheme.

- Fast new flow establishment. Handover must be handled by the resource management scheme in a fast way. It must handle handover rates, which will be at least two per call, and it should be able to perform fast changes such that the loss of packets and the delay are brought to a minimum.
- Edge-to-edge reservation is enough. End-to-end QoS is more than desirable and also too difficult to solve, taking into account the variability of the access networks. In this case, edge-to-edge means that there is only one operator in the network or only a few trustable ones. The resource management scheme is homogenous and there is no need for defining mapping between all kinds of resource management methods. Furthermore, it is not necessary to solve problems with tunneling, encrypted flows or flows through Network Address Translators (NATs) [RFC1631] and firewalls.
- No per-flow states in the core nodes. As much resource management functionality as possible must be in the edge routers and not in the core routers. The edge routers typically have to perform per-flow management as part of the bearer service anyway and hence, complex per-flow handling is not a significant burden. However, core routers can not have per-flow responsibilities for the bearer service. It is recommended to optimize for simple QoS mechanisms in the core and use mechanisms that are more complex only in the edge nodes.
- Unicast is enough. The majority of the traffic in the RAN is unicast transmission of radio frames between the Node Bs and the RNC. Multicast traffic represents only a negligible portion of the traffic in the network. If a multicast resource management scheme is used, it is usually at the expence of the performance of resource management of unicast traffic. So the resource management scheme should be optimized for handling unicast traffic in the network.
- Minimal impact on router performance. The introduction of QoS mechanisms and signaling should not impact the performance of the infrastructure. The routers are already fully loaded and must not have more computation to perform.
- Bi-directional signaling. In the current RAN, the RNC is initiating the flow both to and from the mobile station through some radio base stations. The flows are bi-directional and to set up the flow, including the resource management functionality, must be as fast a possible. Even for a bi-directional flow, only one single loop of signaling messages can be allowed.

Chapter 2 will investigate which resource management schemes exists and how they fulfill these requirements. It will be shown that there is a need for another resource management scheme, which is the RMD scheme introduced in chapter 3.

1.4 Scope of this Thesis

The RMD scheme, introduced in this thesis, consists of two tightly coupled protocols, the per-domain reservation (PDR) protocol and the per-hop reservation (PHR) protocol:

- The PHR protocol is processed in all nodes in the Differentiated Services domain on a hop-by-hop basis. The PHR extends the Differentiated Services per-hop behavior (PHB) by enabling resource reservation.
- The PDR protocol is only used between the edge nodes and represents the resource reservation in the complete Differentiated Services domain. So the PDR functionality is only implemented in the edge nodes.

Currently, only one PHR protocol is defined in RMD, the RMD On-demand Reservation protocol (RODA) [WeJa01b]. In the RODA PHR, resources are requested dynamically and reserved on demand in the nodes of the RMD scheme.

The main goal of this assignment is to first specify, in a detailed way, the architecture and implement the reservation-based PHR and PDR functionality that has to be fulfilled by an RMD node. In particular the objectives of this thesis work are:

- Identify and study different resource management schemes, including Integrated Services, Differentiated Services and load control with respect to the IP-based UTRAN.
- Choose the best scenario that efficiently combines the Integrated Services, Differentiated Services and a lightweight resource management scheme. Furthermore, in the context of the chosen scenario specify the reservation-based RMD on the nodes such that the implementation of this functionality could be easily realized.
- Specify an experimental PDR protocol for a fictitious cellular network scenario.
- Implement the basic parts of the RODA PHR and the experimental PDR using the Linux operating system and perform a basic demonstration using this prototype.

The two first bullets were made in co-operation with the other RMD team members, while the last two were only the work of this thesis.

1.5 Organization of this Thesis

This chapter has introduced the cellular network and the use of IP there. It concluded that a reservation management scheme must be used, since IP does not provide that functionality. Furthermore, it described the scope of the work of this thesis and here it presents the organization of the thesis. Chapter 2 will list some existing resource management schemes that try to solve the missing functionality in IP networks. The chapter includes a basic functional comparison of the schemes and concludes that no existing protocol can be used in the RAN environment. Chapter 3 will describe the RMD scheme including both PHR and PDR protocols. Only the RODA PHR protocol will be introduced. Finallly, in that chapter, the use of RMD in the UTRAN is described.

Chapter 4 talks about a prototype implementation of RMD. First it gives an introduction to the QoS architecture in Linux and then it describes how the RMD was implemented in Linux. Chapter 5 gives some results from the prototype and shows some results from the demonstration. Finally, chapter 6 includes a summary together with conclusions and suggestions for further studies.

[2] RESOURCE MANAGEMENT SCHEMES

Chapter 1 showed that dynamic reservation in IP-based UTRAN with good QoS provisioning and high utilization of available resources is needed. Here, in chapter 2, several different QoS architectures for IP are investigated to find out if they are candidates or not. It shows out that no candidate really solves the entire problem of resource management in UTRAN and therefore chapter 3 will introduce a new QoS management architecture. Last, here in chapter 2, that new architecture will be compared with all the existing and proposed resource management schemes introduced in this chapter. First, the two major QoS architectures Integrated Services and Differentiated Services are explained.

2.1 Integrated Services

The Integrated Services (Intserv) architecture [RFC1633] is specified by the IETF and has been around for some time now. It is based on an explicit mechanism for signaling per-flow QoS requirements to network elements with quantitative reservations.

2.1.1 RSVP Signaling

The resource reservation signaling mechanism typically used in Intserv is the resource reservation protocol (RSVP) [RFC2205], [RFC2210]. RSVP is designed as a dynamic mechanism for explicit reservation of resources for each flow. Figure 2-1 shows how the reservation works for a new flow.

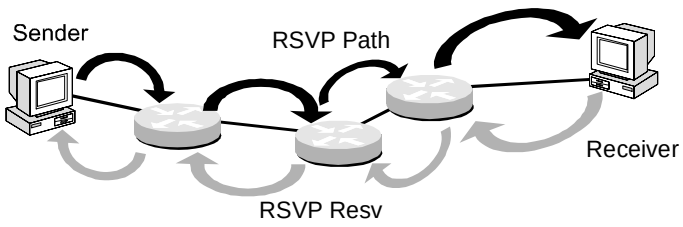

Figure 2-1: Resource reservation for new flow in Intserv using RSVP

The sender will send an RSVP Path message to the receiver. That message contains a description of the traffic characteristics. Every router in the path between the sender and the receiver installs a state for the flow and forwards the message to the next hop. An IP option, called Router Alert [RFC2113], is used to inform each intermediate router that the packet must be processed and not only forwarded. The receiver replies on the RSVP Path message by sending an RSVP Resv message back to the sender to make the reservation. Since each

router installed a state, it is now possible for each router to make sure the RSVP Resv message takes the same way back to the sender. At this time, the reservation is made and traffic can start to flow. This technique is called receiver-initiated, because the receiver is the one who makes the actual reservation and not the sender. Note that the reverse path must be stored by the nodes, because the routing is not always symmetric.

To align this resource reservation better with the connectionless robustness of IP, the state information is maintained through so called soft state principle. With soft state, the state information is periodically refreshed by sending refresh signaling messages. When these periodic refreshments are not received, the state information is automatically deleted after a certain time.

Although Intserv is designed to provide end-to-end QoS, it is currently not widely deployed. The main reason for this is that reserving resources per-flow introduces severe scalability problems at the core nodes, where the number of processed flows is in the order of millions. Therefore, the usage of the Integrated Services architecture is limited to small access networks, where the number of flows using reservations is modest.

2.1.2 Token Bucket

To specify the traffic characteristics, the Intserv model uses the concept of a token bucket. A token bucket is a First In First Out (FIFO) queue with a certain drain rate as depicted in Figure 2-2.

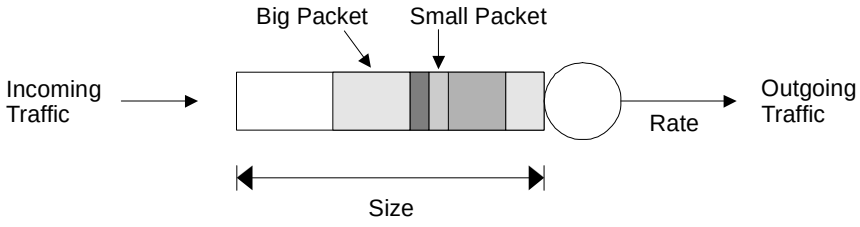

Figure 2-2: Token bucket

In the Intserv model, the token bucket is specified by five parameters:

- Rate. Specifies how fast the queue is emptied in bytes per second.
- Size. Specifies the queue length in bytes.
- Peak data rate. Specifies the absolute maximum rate in bytes per second.
- Minimum policed unit. Due to the limitations in the link layer protocols, it is common that small packets must be padded with zeros up to a certain size. This value specifies how small a packet can be in bytes, without being padded. If a packet is smaller than this value, it will still be treated as if it was of this size.
- Maximum packet size. The largest size of one single packet, specified in bytes. Packets are never allowed to be bigger, or must be fragmented into smaller packets.

A flow is conformant with the token bucket specification if it can be shaped by it without making the queue full. The average rate of a long lasting flow must always be equal to the token bucket rate or less. If the token bucket size is big, it means that a bigger variation in the rate is tolerated. Peaks can be accepted as long as they have a limited duration time and the

rate goes back to normal after some short time. This traffic model is good for specifying flows with some variations in the rate, such as MPEG video streams.

Policing is the process of making a flow conformant to a token bucket specifier. This means that packets must be discarded in order to make it conformant. A policer can be implemented by simulating a token bucket as in Figure 2-2. If a packet is bigger than the maximum packet size or if a packet arrives when the buffer is full, then it is discarded. Packets are also discarded if they exceed the peak data rate.

Shaping is the process of taking a flow with variable bit rate, but still conformant, and removing the peaks above the token bucket rate. A shaper is typically implemented with a FIFO as above where no outgoing traffic exceeds the token bucket rate. When a peak arrives, some packets are queued for later delivery.

2.1.3 Intserv Node Architecture

Consider a single RSVP- and Intserv-aware node with two interfaces. In the Figure 2-3, all flows go from left to right. The sender sends an RSVP Path message. The node will install a state for this new flow in the control plane. The usual IP routing module will be used to find the outgoing interface for the flow. Later, an RSVP Resv message will arrive from the receiver and the node will at that time update the state in the control plane. It will also change the packet classifier and construct a policer and shaper in the data plane for the new flow.

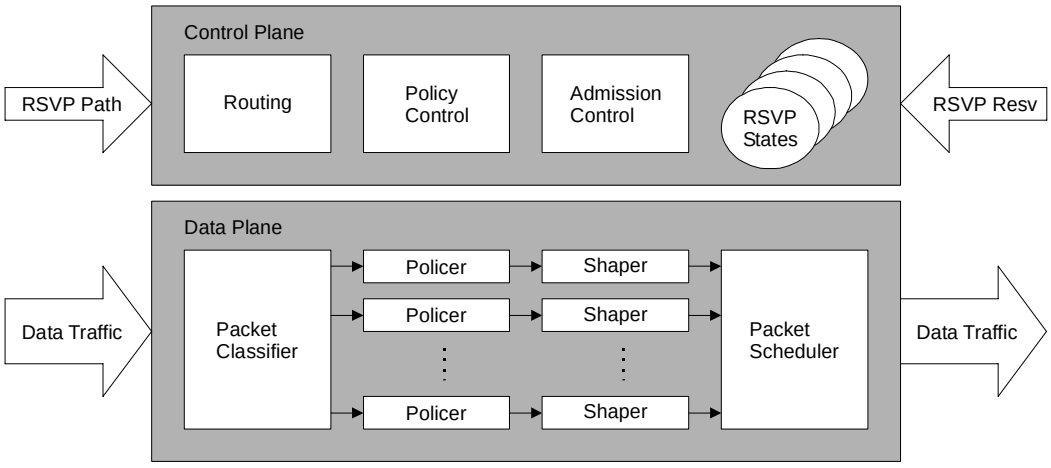

Figure 2-3: Intserv node architecture

The more flows the node must process simultaneously, the more work it has to do in both the data plane and the control plane. Every single data packet must be classified into its correspondent flow and then policed and shaped if necessary when using Intserv. In a core node, this is just too much work. The core routers are already overloaded and introducing per-flow treatment will seriously degrade the network performance of the router, especially when the number of flows in the core network increases. In the access network, near the end-users, the number of flows are moderate, but in the core, all these flows will be aggregated into too many flows. This problem led to the initialization of Differentiated Services.

2.2 Differentiated Services

The Differentiated Services (Diffserv) architecture [RFC2475], [RFC2836] was introduced as a result of efforts to avoid the scalability and complexity problems of Intserv. Scalability is achieved by offering services on an aggregate basis rather than per-flow and by pushing the per-flow states to the edge of the network. The service differentiation is achieved using the six highest bits of the Type of Service (TOS) field in the IP header. These bits are referred to as the Differentiated Services codepoint, or DSCP. This value is used to differentiate between different kinds of traffic. For example, a network operator may assign value X for real-time voice over IP and value Y for bulk transfers. Then data packets with DSCP set to X is given higher priority than all other traffic and data packets with DSCP Y is given the lowest possible priority in every node.

To support different traffic types, each node need to behave in the same manner for each traffic type. Therefore, a set of behaviors is specified. These behaviors are called per-hop behaviors (PHBs) and there are currently two PHBs specified: Assured Forwarding (AF) [RFC2597] and Expedited Forwarding (EF) [RFC2598]. An operator that wants to support a certain PHB must assign a DSCP value for the PHB and then upgrade and configure all his nodes to support the PHB.

Diffserv defines the concept of a Differentiated Services domain. One operators network can be a Diffserv domain, or he can divide his network into several non-overlapping domains. A domain is a connected sub-network, such as the one depicted in Figure 2-4. The traffic inside the domain is considered as trusted by its operator, while traffic entering the domain is not. The operator makes sure that no external customer or operator can misuse the resources of the domain. This is achieved by policing and shaping at the edge nodes. The edge nodes are routers with a connection to another node outside the domain and will perform policing, shaping and marking on traffic entering the domain. The non-edge nodes, which are called interior nodes, only perform packet scheduling based on the marking done by the edge nodes.

Assume that the left host in Figure 2-4 is the sender and the right host is the receiver of a flow. For this flow, the left edge node will become the ingress node where the flow data traffic enters the domain. At that node, it is necessary to perform marking, policing and shaping. The right node will be the egress node for the same flow and there it might only be necessary to perform remarking and maybe shaping. What node is ingress and what is egress depends always on which flow is under consideration. An edge node can act as both ingress and egress at the same time for different flows and therefore must all edge nodes be able to perform marking, policing and shaping.

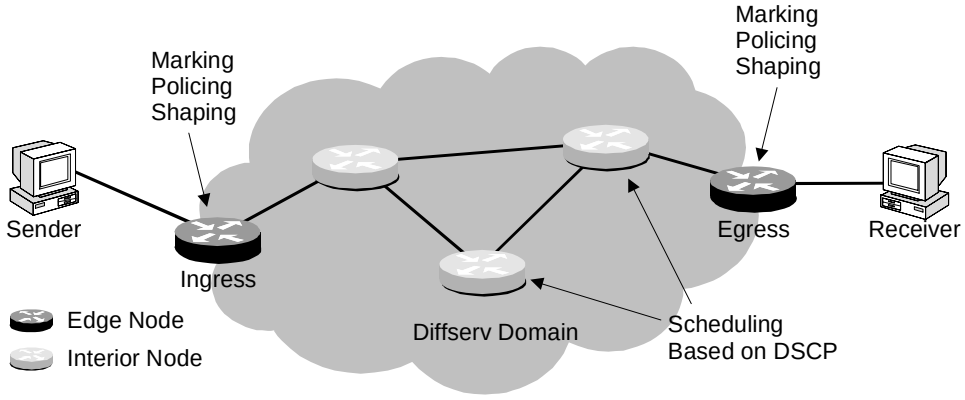

Figure 2-4: A Differentiated Services domain

The operator will provide different services to its customers by having a service level agreement (SLA) with the customers. The SLA specifies what services are available, including costs, and what the customer must do to access the different services. An operator may have a premium traffic class, which will receive the highest possible priority. In order to access this service, the customer must mark his premium traffic in a specific way and also pay a higher price for the premium service. See [Kilkki99] for more examples of services that an operator can offer in a Differentiated Services network.

Diffserv is promising but there are a lot of issues that have to be solved. Diffserv surely solves the scalability problem of Intserv, since the core nodes only need to perform scheduling based on the DSCP. The theoretical maximum number of DSCPs is 64, while the number flows might be much, much more. However, Diffserv solves this at the cost of quantitative QoS. It is not possible to reserve resources in a Diffserv domain since no resource management system is specified by Diffserv. For instance, when a customer uses the premium traffic class mentioned above, they expect very low delay and few lost packets. However, if too many customers use the service at the same time, there might be a congestion somewhere in the Diffserv domain and the customer will experience a degraded service with lots of lost packets. Either this problem must be solved by other means, such as over-provisioning, policing or pricing, or by using some kind of resource management system. Diffserv is just a framework for differentiation between different kinds of traffic classes and a few per-hop behaviors.

2.3 More existing QoS architectures

The two most important QoS architectures were just described and here a few more will be briefly described. In section 2.4, the QoS architectures interesting for the UTRAN will be described.

2.3.1 Over-provisioning

One way to overcome the limitations of the best effort model networks in providing QoS, is by over-provisioning the networks. A good service can be achieved by allocating more bandwidth than the expected network peak requires. Although over-provisioning of the network increases the probability of having enough resources available for real-time applications. It does not guarantee the quality of service for these applications and on the other hand, there will always be a waste of resources in an over-provisioned network.

Over-provisioning can also be used in Diffserv to solve the problem with the premium class traffic mentioned in section 2.2. By giving premium traffic at least 1 Mbps in every interior node and only offering 10 kbps of premium service to each customer, the operator can achieve good guarantees if the number of customers is less than 100. Each customer can only send 10 kpbs of premium traffic, the policer at the edge nodes will drop the rest. This will give a guarantee that the aggregated traffic in any interior node will never exceed 1 Mbps and hence give the expected QoS.

2.3.2 Intserv over Diffserv

This scheme [RFC2998] can provide end-to-end QoS for applications by using the Integrated Services model over Diffserv domains. The network consists of several Diffserv domains, which the Intserv reservations will cross. It includes some combination of Intserv nodes, which does per-flow classification plus admission control, and Diffserv nodes, which does only aggregate traffic control per PHB. Individual routers may or may not participate in RSVP signaling, regardless of where in the network they reside.

One problem that immediately arises when you combine Intserv and Diffserv is how to do resource management in the Diffserv domain. The end user does a quantitative reservation using Intserv and expects this reservation to be guaranteed also in the intermediate Diffserv domain, which is a service that the Diffserv architecture does not offer. In [RFC2998] three possible solutions are discussed:

- Static or over-provisioning. The amount of traffic, each edge node will allowed into the network is configured statically by the operator. The main drawback of this architecture is the high level of resource over-provisioning that has to be used, as discussed in the previous section.
- Dynamic provisioning using RSVP. With this method RSVP packets are sent into the network and some strategic interior nodes are RSVP-aware. These nodes will perform admission control using RSVP only in the control plane, but they will still not use RSVP per-flow scheduling in the data plane. This will significantly help the scalability problem of original RSVP since data packets can be handled per behavior aggregate. Nevertheless, the problem with states per-flow still exists. This problem can further be solved by the use of aggregation of RSVP flows, as described in section 2.4.1.
- Dynamic provisioning using something other than RSVP. A bandwidth broker [ReOn98] or another signaling protocol can be examples of alternatives to RSVP.

2.3.3 Load Control

Load Control [WeTu00] is a scheme that tries to fill the missing gap of resource management in Intserv over Diffserv. The Load Control scheme does this without using any extra signaling packets or having any per-flow states in the core routers. The Load Control information is carried in the IP packet header by using only two bits. One bit is for making reservations and the other one is for the admission control result. If an interior node cannot admit a reservation, it sets the second bit in order to inform the edges about the situation. This is called marking. The Load Control scheme specifies two modes of operation:

- Simple marking. Which is a measurement–based admission scheme where the routers measure the traffic volume and base the marking on these results.
- Unit-based reservation. This scheme can be used to perform resource reservation of the amount of traffic that is transmitted by allowing the sources to keep their reservations independently of the volume of the actual traffic transmitted. In this scheme, the router uses the first bit to request and maintain the reservations for the flows. Each flow can occupy a certain number of units of resources, which are a share of the bandwidth that can be reserved by the edge nodes.

The load control scheme is very simple and scalable, but it lacks a lot of functionality. In the form specified in [WeTu00], it cannot be used in a general Intserv over Diffserv environment. It must be extended with many important features. RMD, specified in section 3, is such an extension.

2.4 Proposed Lightweight Resource Reservation Schemes

Currently, there are several resource reservation protocols that are good enough to be seen as possible candidates for dynamic provisioning of resources in the IP-based UTRAN. This section briefly describes some possible resource reservation proposals.

2.4.1 RSVP Aggregation

One way of solving the scalability problem of Intserv is to aggregate end-to-end flows in the core. The RSVP Aggregation scheme [BaIt01] defines an aggregation region. Aggregation of end-to-end flows is performed inside the region. Figure 2-5 shows an aggregation region where four end-to-end flows are aggregated into only two aggregated flows within the region.

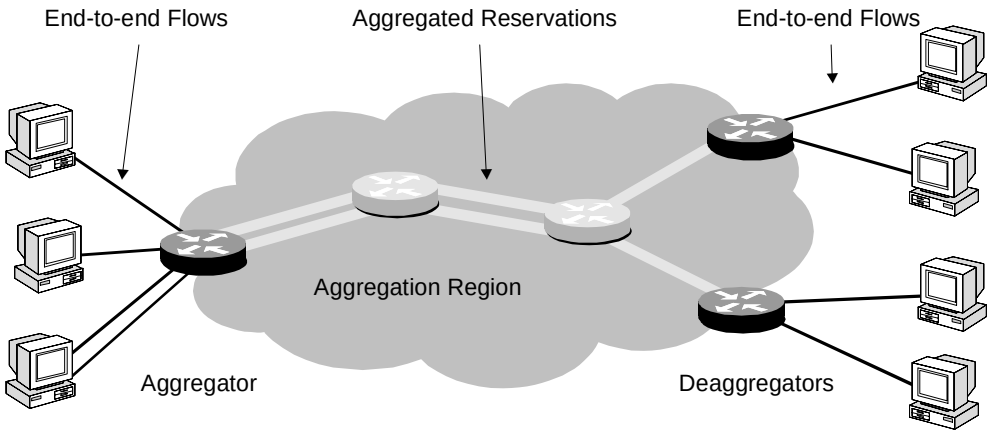

Figure 2-5: An RSVP Aggregation region

The first router in the aggregation region that handles the aggregated reservations is called the aggregator, while the last router in the aggregation region that handles the reservations is called the deaggregator. The interior nodes in an aggregation region must also be RSVP-aware, since the aggregated reservations also are made with RSVP. When a new end-to-end flow arrives at the region border, the aggregator will place the end-to-end flow into the aggregated reservation that goes to the correct destination. If that the aggregated reservation does not have enough reserved resources then the aggregator will update the reservation with some extra resources using ordinary RSVP messages inside the region. At the same time, all RSVP messages for the end-to-end flows are ignored by all interior nodes in the aggregation region.

The RSVP Aggregation scheme will decrease the number of states in all interior nodes. The number of states is only dependent on the number of aggregated flows, but the packets still need to be classified into one of the aggregated flows at each node. Combining RSVP Aggregation and Intserv over Diffserv can solve that problem. In that case, the number of states will be the number of aggregated flows, and packets must only be classified on the DSCP. This is of course a very good solution but it still has some major problems. RSVP is a very complex protocol and aggregation is not a simple task, which makes RSVP Aggregation even more complex. However, the main disadvantage with the RSVP Aggregation is that even if the flows are aggregated, the interior nodes will have to support a high number of aggregated RSVP states if the region is big. If, for example, an aggregated region has N edge nodes, then the number of aggregated RSVP states that an interior node will have to maintain is in the order of N^2. If the regions are small, the gain is not so big since the number of nodes in the path that will act as aggregators and deaggregators will increase.

2.4.2 Boomerang

Another way to make a scalable solution is to make the signaling protocol simpler. Boomerang, which is presented in [FeNe99], [BeCs00] and [AhBe99], is such an approach.

The major difference between RSVP and Boomerang is that Boomerang is sender-initiated, while RSVP is receiver-initiated. This means that the path messages used in RSVP are not necessary in boomerang. Instead, only one single loop of signaling packets is needed per reservation. Otherwise, Boomerang is very similar to RSVP and can also be used as an end-to-end protocol, or in an Intserv over Diffserv environment. Furthermore, it is also possible to define a boomerang aggregation scheme in a similar way as with RSVP Aggregation.

2.4.3 YESSIR

YESSIR [PaSc98] is another sender-initiated end-to-end reservation scheme. Instead of defining new signaling packets, it extends the real-time protocol (RTP) [RFC1889] with resource reservation. It is assumed that all applications that need to reserve bandwidth also use RTP and its control packets. YESSIR extends the control packets with extra resource reservation information and suggests to use the IP router alert option [RFC2113] in the same way as RSVP does. Otherwise, YESSIR provides much of the functionality of RSVP, but with a significantly reduced protocol overhead. The most important difference with RSVP is that YESSIR is sender-initiated instead of receiver-initiated.

2.4.4 Dynamic Reservation Protocol

Dynamic Reservation Protocol (DRP) is presented in [WhCr98]. This protocol is also a sender-initiated reservation-based protocol for end-to-end use, but it is optimized for large multicast groups with many senders and receivers. It will merge signaling messages from different senders in a multicast group. However, DRP still needs to store the reverse routing information just as RSVP. So for unicast traffic, DRP has more or less the same problems as RSVP.

2.4.5 Dynamic Packet States

Dynamic packet states (DPS) [StZh99] is an extension to Diffserv. The idea of this method is to specify a resource management protocol for use only in a Diffserv domain. The DPS can for instance be used in a Diffserv domain offering Intserv services using the Intserv over Diffserv model described in section 2.3.2. Therefore, DPS is not a protocol that can be used end-to-end and therefore it can not substitute RSVP. We call these kinds of protocols edge-to-edge, since they operate between edge nodes in a Diffserv domain.

With DPS, each packet carries some PHB-specific state in its header in addition to the DSCP value. This state is initiated by the ingress node. Any interior node that receives such a packet processes the packet, based on the state carried in the packet's header. It updates both its internal state and the state in the packet's header before forwarding it to the next hop. However, DPS is not a complete protocol but only a framework for designing these kind of protocols.

2.4.6 Scalable Resource Reservation Protocol

Scalable Resource Reservation Protocol [AlFe98], or SRP, is a measurement-based admission control (MBAC) protocol. Measurement-based means that when a sender wants to initiate a flow it simply starts to send the data, but marked with a request flag. Before the intermediate nodes forward a request marked packet, they do a simple admission decision. If they forward the packet, it means they can handle the extra traffic defined by the packet. They learn the requested rate by measuring the rate of request marked packets and possibly drop some portion of them, depending on how much free resources they have. The receiver, using an estimator, calculates the incoming rate of the request packets and reports that to the sender. The sender may then continue to send data packets, but then marked with a reserved flag.

Those packets keep the reservation in routers in a soft state manner. No per-flow states are stored in the intermediate nodes, only traffic monitoring must be done on each network interface.

SRP is specified as an end-to-end protocol, but this demands that the end-hosts are trustable. Therefore, SRP can not be used as an end-to-end protocol, but it is of course possible to use SRP in an edge-to-edge manner in a Differentiated Services domain, since the edge nodes can be trusted. Other edge-to-edge measurement-based protocols are [BiBl01] and FC-DS.

2.4.7 Feedback Control Extensions to Differentiated Services

This scheme, also called FC-DS, is described in [ChLe99] and is a measurement-based admission control extension to Differentiated Services. The measurement approach used here is a bit different from SRP. FC-DS still needs a traffic monitor in each interior node, but instead of using actual data traffic for probing the status of the network, FC-DS sends probe packets into the domain. An edge node generates a probe packet, sends it into the domain and asks each interior node for its available resources. This information is then used when admitting new flows into the domain.

[ChLe99] specifies several methods on when to generate probe packets. One method is to generate such a probe as soon as a new request arrives and for that particular request. When the response is received, the admission control decision is taken. Another possibility is to periodically probe the domain. Every node sends probe packets per PHB class and per edge node once every time period. By collecting this information, an immediate admission control decision can be made.

By making the time period short enough, you will get a good resource management architecture with smaller scalability problems. The scalability problem arises if you have a network with many edge nodes and still want to have a short time period between the probe packets.

2.4.8 Distributed Connection Acceptance Control

This scheme is presented in [GiKe99] and it describes how a pricing mechanism can be used by a packet network to carry a telephony-like service with low packet loss. It is a sender-initiated measurement-based admission control scheme that is marking packets when the resources are highly utilized and then charging a fixed small amount for each mark. In the nodes used in the core network, the marking strategy defines a threshold in such a way that the packets are marked when this threshold is reached. The higher the selected threshold, the higher is the price that the user has to pay. This is because the admitted number of requests is directly proportional to the selected threshold.

2.4.9 Ticket

This protocol is described in [Erik98]. The Ticket protocol is sender initiated and operates on an end-to-end basis. The per-flow states are kept at the end-hosts, while the interior nodes in the connectionless network are aggregated states on either per-priority or per-link basis.

When an application needs to initiate a flow with a controlled QoS, the sender sends a message to the network with a request for a specific traffic contract. In this scheme, an ingress node, denoted access router, will receive this request. The request travels through the access node and through all the intermediate nodes towards the receiver. On each node's output link, this request is subjected to admission control. If the admission control is successful, the request will reach the receiver, otherwise it will be dropped. The receiver returns the request to the access router. The access router recognizes that the request is

accepted and translates it into a so-called ticket message that contains all data about the traffic contract. This message is then sent back to the sender. The information in the ticket message is protected from forging with a digital signature. Periodically the sender transmits the ticket message to the network by inserting it in the user data packet flow. In this way, the nodes in the network can use the ticket message to extract all the information that is needed about the traffic contract of the connection. These include bandwidth, priority level, other QoS parameters, expire time and so on.

Since the ticket message contains all traffic contract parameters, no per-flow states need to be stored in the network. Instead, each node counts all the received ticket messages it and uses that information when making admission decisions for new flows. The only exception is that the access node still needs to have per-flow states.

2.5 Comparison of the Resource Reservation Schemes

In section 1.3 the RAN requirements on the IP-based UTRAN are given. This comparison includes the resource management protocols described in section 2.4 and RMD, which is specified in the next chapter. The comparison is made using the following features:

- Reservation-based. Does the protocol use on-demand reservations to guarantee the quality? This scheme usually uses something similar to Intserv or a token bucket to specify how the traffic characteristics of the new flow is.
- Measurement-based. Does the protocol make use of a measurement-based approach? The flows must not change their characteristics too much, since the approach is based on the fact that flows do not change too much.
- Fast new flow establishment. Is it possible to set up a new flow using only one single loop of signal messages or faster?
- End-to-end protocol. Can the protocol be used as an end-to-end protocol, including security matters, or is it limited to use in a Diffserv domain or similar?
- Per-flow states in the core nodes. Is it necessary to store per-flow information in core nodes?
- Sender initiated. Is the reservation sender initiated or receiver initiated? This feature does not apply on measurement-based schemes.
- Multicast support. Does the scheme have support for multicast? Can different receivers reserve different amount of resources in the case of a reservation-based scheme?
- Bi-directional signaling. Is the signaling bi-directional? Can the admission control be performed in both directions at the same time?

The comparison result is shown in Table 2-1. Each protocol can have one of the following values per feature:

- Supported (Y)
- Partially supported (P)
- Not supported (N)
- Unknown

	Reservation-based	Measurement-based	Fast New Flow Establishment	End-to-end Protocol	Per-flow States in Core Nodes	Sender-initiated	Multicast Support	Bi-directional Reservations
RSVP Aggregation	Y	N	N	Y	P	N	Y	N
Boomerang	Y	N	Y	Y	P	Y	P	Y
YESSIR	Y	N	Y	Y	Y	Y	Y	N
DRP	Y	N	P	Y	Y	Y	Y	N
DPS				N	N	Y		
SRP	N	Y	N	P	N	Y	Y	N
FC-DS	N	Y	Y	N	N	Y	N	N
Distr. C.A.C.	N	Y		Y	N	Y	N	N
Ticket	N	Y	Y	Y	N	Y	N	N
RMD	Y	Y	Y	N	N	Y	N	P

Table 2-1: Comparison evaluation

Based on this comparison we can conclude that none of the compared resource reservation protocols in section 2.4 satisfies the majority of the requirements presented in section 1.3. Remember that a resource reservation scheme for IP-based UTRANs must offer very high quality QoS and at the same time be scalable and use the network resources efficiently. The flow establishment must be very fast, with support for bi-directional reservation. Furthermore, it must be very simple and only make a small impact on the core routers. However, on the other hand, it does not need to support multicast, nor must it be an end-to-end protocol. Therefore, one can deduce that in the IP-based UTRAN a new resource reservation protocol must be developed.

[3] THE RESOURCE MANAGEMENT IN DIFFSERV SCHEME

The Resource Management in Diffserv (RMD) scheme [WeJa01a] was developed to solve the problem of resource reservations in UTRAN. In the previous section, it was described why existing resource reservation schemes could not be used in the RAN environment. The purpose of RMD is however not limited only to UTRAN, but it is also a scalable resource management scheme for any IP-based network that is using Differentiated Services. RMD extends the Diffserv architecture with resource reservation concepts and features. It is a framework of two protocols, which extend a Diffserv domain with resource reservation. These two protocols are called the per-hop reservation (PHR) protocol and the per-domain reservation (PDR) protocol.

This chapter describes the RMD scheme by first outlining the framework and then describing what the PHR and PDR protocols do. After that the RMD On-demand (RODA) PHR protocol is introduced and then, an outline of a PDR protocol is given. Finally, the use of RMD in UTRAN is discussed.

3.1 Resource Management in Diffserv Framework

The RMD framework proposal is based on standardized Diffserv principles for traffic differentiating and extends these principles with new ones necessary to provide resource management and admission control in Diffserv domains. It is assumed that some kind of external mechanism exists for signaling reservation requests to the edge nodes in the Diffserv domain. This mechanism can be quantitative reservation-based QoS signaling protocols similar to the protocols described in the chapter 2. The edges use RMD to decide if the request can be admitted. For this purpose, the RMD framework defines two major protocol concepts, the PHR and PDR. See Figure 3-1. The PHR protocol is used to either install reservation states or monitor available resources in each interior node in a communication path, while the PDR is used for exchanging information between the edge nodes and is completely ignored by the interior nodes. Furthermore, the PDR protocol is used for tunneling the external QoS request through the domain, for passing resource management information between the edge nodes.

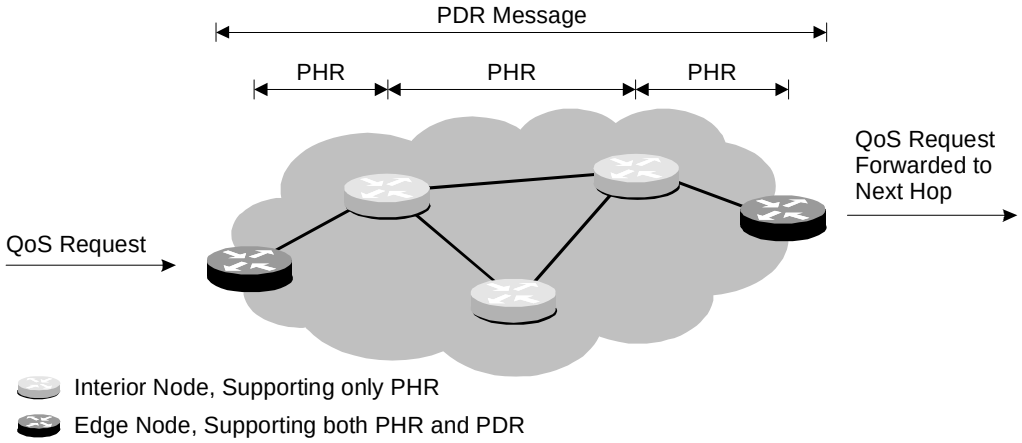

Figure 3-1: Per-hop reservation (PHR) and per-domain reservation (PDR) protocols

The PHR protocol extends the PHB in Diffserv with resource reservation, enabling reservation of resources per traffic class in each node within a Diffserv domain. The traffic class is defined by a DSCP value or a set of DSCP values, as in the AF per-hop behavior classes. AF1x is one such class and consists of the DSCP values 10, 12 and 14. The reservations can be performed for each DSCP value independently or all of them together. The operator can decide how the reservations are performed. Furthermore, the operator may also decide not to use RMD in some of the traffic classes, say in AF2x, and for instance use static provisioning there instead. In the best effort traffic class, the operator will typically not use any resource management at all. However, the PHR is not able to differentiate between different traffic flows and this is a strength, since no per-flow information is stored and no per-flow packet scheduling is necessary. This also makes it impossible to have a receiver-initiated PHR protocol, since no backward routing information is stored in the interior nodes. If the external protocol is receiver-initiated, this must be handled by the PDR protocol instead. The PHR works only with traffic classes and not with per-flow states. Therefore, it scales very well to large Diffserv domains.

Currently, one PHR protocol is specified, the RMD On-demand (RODA) PHR protocol. This scheme [WeJa01b] can be used to perform resource reservation of the amount of traffic that is transmitted. The edge node uses certain packet types to request and maintain the reserved resources for the flows. Each flow can occupy a certain number of resource units, which is a bandwidth parameter that must be reserved by all interior nodes in the communication path.

However, it is possible to define a measurement–based PHR protocol, where the nodes measure the traffic volume and base the admission or rejection of the resource reservation requests on these results. Such a protocol can use special packet types to probe for available resources in the interior nodes.

In this thesis, only the RODA PHR is investigated. When using the RODA PHR, the resources are requested dynamically per DSCP and reserved on-demand in the interior nodes. The per-flow functionality is kept at the edges while the interior nodes keep only aggregated states per DSCP. The reservation is done in terms of resource units, which may be based on a single parameter such as the bandwidth or on more sophisticated parameters, such as effective bandwidth [GiKe97].

The PDR protocol manages the reservation of the resources in the entire Diffserv domain and is only implemented in the edge nodes of the domain. This protocol handles the interoperation with the external resource reservation protocols. The PDR protocol is the link

between the external resource reservation schemes and the PHR protocol. In the context of the RMD, it is possible for the PDR to inter-operate with one or more PHR protocols. Furthermore, the PDR should always be

able to interpret the external resource requests and map them into an appropriate DSCP and the parameters for the used PHR protocol. Moreover, the PDR protocol should also implement all the features available in the external protocol.

3.1.1 PHR Signaling Message Types

All PHR messages are generated by the ingress node and sent in the direction of the far-end receiving host. The message must never leave the Diffserv domain and is therefore discarded by the egress node. The following messages are two examples of messages that can be defined by a PHR protocol:

- PHR_Resource_Request. It is generated by the ingress node in order to initiate or update the aggregated soft state reservation in the communication path to the egress node. In case of a measurement-based PHR protocol, it will check the traffic load status in each node in the path.
- PHR_Refresh_Update. This signaling message is specific to reservation-based PHR protocols. It is generated by the ingress node to refresh the soft state reservation in the communication path to the egress node.

It is important that PHR messages are not lost in the Diffserv domain and therefore, it is recommended to give them higher priority then usual data packets. If possible, the interior nodes should start to mark the PHR_Resource_Request messages before they start to mark the PHR_Refresh_Update messages. The PHR_Refresh_Update is generated because a soft state reservation needs to be refreshed and should be in favor to a PHR_Resource_Request, which represents a new flow. It is better to reject new incoming requests than to tear down already existing flows.

3.1.2 PDR Signaling Message Types

The PDR signaling messages are processed only by the RMD edge nodes and not by the interior nodes. Either the PDR protocol can be an entirely new protocol or it may use an existing one such as RSVP, Common Open Policy Service (COPS) [RFC2748], etc. In this thesis, a new protocol will be used. A list of possible PDR messages is given below. Note that they are messages and not necessary IP packets on their own. It is likely that a PDR message is sent together with a PHR message in one single IP packet. The following list is just examples of PDR messages that can be defined by a PDR protocol:

- PDR_Reservation_Request. This message is generated by the ingress node in order to initiate or update the PDR state in the egress node. This PDR signaling message is typically sent together with a PHR_Resource_Request message. It may also contain the flow identifier used by the external protocol and its QoS parameters, so that the QoS request can be re-generated at the egress node and sent to the next hop.
- PDR_Refresh_Request. The ingress node sends this message to the egress node to refresh the PDR states located in the egress node. In case a reservation-based PHR is used, the PDR signaling message should be sent together with the PHR_Refresh_Update message.

- PDR_Release_Request. This message can be used if the external protocol uses release messages. Many resource reservation protocols, including all protocols that do not use a soft-state reservation principle, usually use release messages to release reservations.
- PDR_Reservation_Report. This report message is sent by the egress node back to the ingress node to report that an IP packet containing a PHR_Resource_Request message and/or a PDR_Reservation_Request message has been received. Furthermore, this message reports the status of the PHR_Resource_Request and the reservation to the ingress node.
- PDR_Refresh_Report. This message is similar to the PDR_Reservation_Report message, but is used to report the status of the PHR_Refresh_Update and PDR_Refresh_Request messages instead.
- PDR_Release_Report. This is sent by the egress node to the ingress node to report that a PDR_Release_Request message has been received and has been processed.

Some of the PDR messages are sent together with a PHR message. However, it is also possible that a PDR message is sent by its own. Therefore, it is important that PDR-only packets also get a higher priority then usual data packets.

3.1.3 RMD Normal Operation

Normal operation refers to the situation when no problems are occurring in the network, such as route or link failure, severe congestion, loss of signaling messages, etc. In Figure 3-2, an example is given that demonstrates how RMD operates under normal conditions, using a reservation-based PHR, such as the RODA PHR. In this example, it is assumed that a sender wants to reserve resources for a new flow using a sender-initiated end-to-end resource reservation scheme. The request is reservation-based and contains such information as a flow identifier and quantitative QoS parameters. The flow identifier is typically the destination IP-address and the destination port number used by all packets in the flow. The QoS parameters can be a token bucket specifier, containing parameters such as data bit rate and maximum packet size, see section 2.1.2. Figure 3-2 shows what happens when such a reservation requests enters a Diffserv domain that uses the RMD framework.

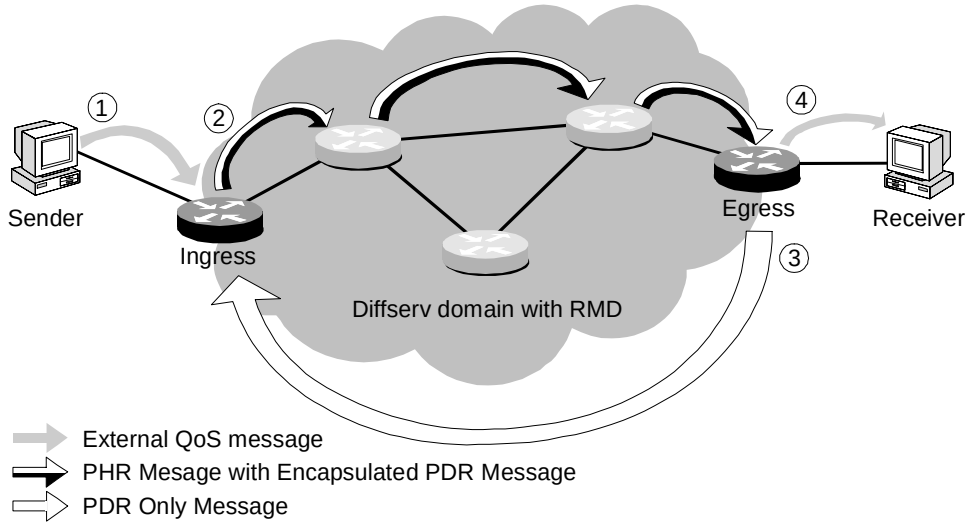

Figure 3-2: Message interaction in RMD, normal operation

In the first step, an external end-to-end QoS request arrives at the ingress node. The ingress classifies the request into an appropriate traffic class by giving the flow a DSCP value. The ingress can then do a local per-flow admission control based on the external protocol mechanism or based on a SLA. After that, it needs to translate the QoS parameters in the external request into the parameters used by the PHR protocol, such as the bandwidth requirements. The flow identifier will be extracted from the external request and together with the reservation state, it will be stored in the ingress, so that the ingress can keep track of which flows are admitted.

If the QoS request is satisfied locally in the ingress node, then the second step is to generate both a PHR_Resource_Request message and a PDR_Reservation_Request message. The PDR_Reservation_Request will typically be sent together with the PHR_Resource_Request message in one single IP packet. The IP header will have the same values as the data packets for the flow, so that they will follow exactly the same path as the data flow. This means that the source address is the IP address of the far-end sender and the destination address is the IP address of the far-end receiver. The DSCP of the IP packet is the same as the traffic class the flow is associated with.

Along the path, all interior nodes will process the PHR_Resource_Request. They must identify the DSCP from the IP header and, if possible, reserve the requested resources carried in the PHR_Resource_Request. The node reserves the requested resources by adding the requested amount to the total amount of reserved resources for the particular DSCP. The PDR message in the IP packet is ignored by all interior nodes.

Finally, when the PHR and PDR messages arrive at the egress node, both are decapsulated and processed. The egress node discards them and does not forward them out of the domain, although the destination address is the far-end receiver. At this point, step three, the egress node can do a local per-flow admission control based on the external protocol mechanism. The egress can also use the PHR functionality if necessary. If the egress node accepts the new flow, then a new PDR state is installed, including the flow identifier, and then a PDR_Reservation_Report message is sent back to the ingress node to inform about the successful reservation. Finally, the external request is re-created from the information in the PDR_Reservation_Request and sent to the next hop.

When the PDR_Reservation_Request reaches the ingress node, it opens up the domain for the new flow. All traffic that matches the flow identifier is mapped into the assigned DSCP and receives the expected behavior by all interior nodes. The reservation makes sure the network is never over-loaded and to be even more certain, each flow should be policed and shaped by the ingress node according to the reservation, so that no one can over-load the network.

The RMD uses soft states and therefore it is necessary to refresh all reservations in the domain. The ingress node will generate the PDR_Refresh_Request messages in order to refresh the PDR soft state in the egress node and a PHR_Refresh_Update is generated to refresh the PHR soft state in all nodes in the path. The process of refreshing is the same as making a new reservation, with the exception of the external QoS requests. A timer will be used by the ingress to specify when the reservations must be refreshed and the egress node will only send back a PDR_Refresh_Report to the ingress node. No external protocol packet will be generated by the egress node. The PHR resources in any node are released if there are no PHR_Refresh_Update messages received during a refresh period. Figure 3-3 shows the whole process of admission control of a new QoS request and refreshing of reservations.

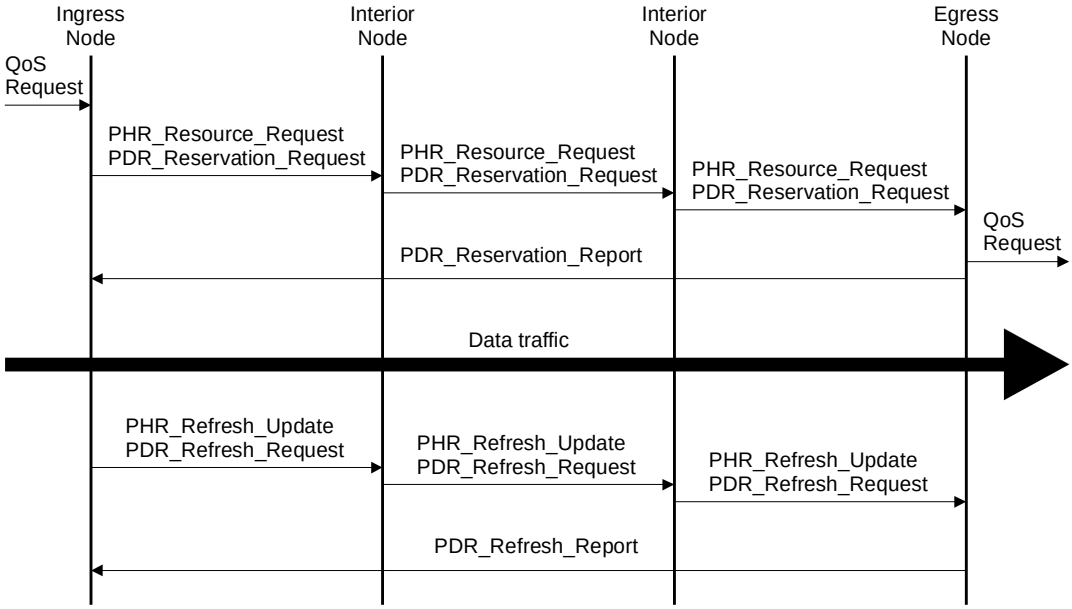

Figure 3-3: Normal Operation for successful reservation

In the next example, one of the interior nodes does not have available resources and cannot admit one extra flow. When the interior node receives a PHR_Resource_Request message, it checks the resource parameters and tries to reserve the requested amount of resources. If that is not possible, the interior node will mark the PHR message and forward it to the next hop. A marked PHR_Resource_Request message is ignored by all interior nodes and just forwarded to the egress node. On the other hand, the PHR_Refresh_Update is processed even when it is marked. When the marked PHR_Resource_Request message reaches the egress node, the egress can see that the reservation failed and does not install any new reservation. It will instead send a marked PDR_Reservation_Report back to the ingress to inform about the failed reservation. Then, the ingress may inform the sender about the rejection using the external protocol. Figure 3-4 shows the process of a flow that is rejected by an interior node.

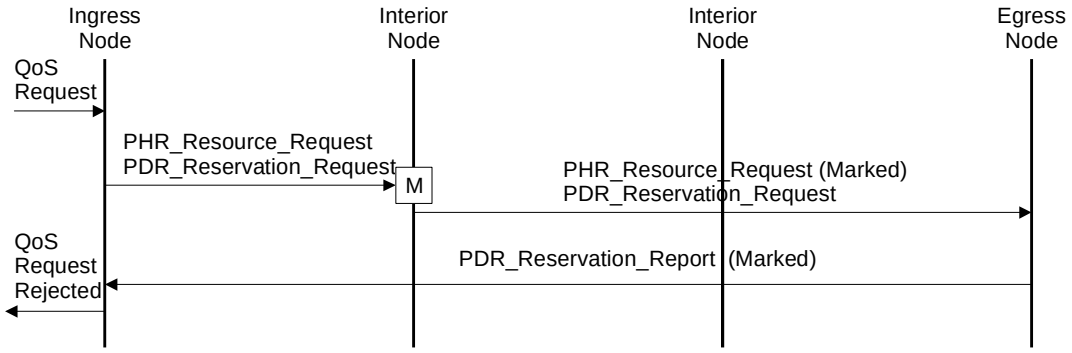

Figure 3-4: Resources unavailable at an interior node

3.1.4 Loss of PHR Signaling Messages

The PHR signaling messages and subsequently the PDR signaling messages might be dropped, for example due to route or link failure. The loss of PHR signaling messages is especially problematic for the reservation-based PHR, since the dropped signaling messages might have reserved resources in some interior nodes in the communication path before they are dropped. These reservations will be occupied until they are released by the soft state timeout and will not be used in the mean time.

The ingress nodes are responsible for handling the loss of the PHR signaling messages. After sending a PHR_Resource_Request message together with a PDR_Reservation_Request, the ingress node will start a timer. The ingress node will then wait for a pre-defined amount of time to receive the PDR_Reservation_Report message. If the ingress node does not receive this acknowledgment within the pre-defined time, it will conclude that an error has occurred. The ingress node will not send any new PDR and PHR signaling messages associated with the same flow during the first subsequent refresh period. In this way, all the possible unused reserved resources will implicitly be released within one refresh period. If the problem is a route or link failure, a re-transmission of the reservation message is likely to fail again.

When a PHR_Refresh_Update message is dropped, the ingress node can do one of two things. It may either continue as if nothing happened or it may terminate the flow. In the first case, the application may experience a possible QoS degradation during one refresh period. If the lost acknowledgment is because of a bit error, or a very temporary failure, then the degradation may be very small or negligible. However, since PHR messages have higher priority in the Diffserv network, a loss means a very serious problem and it is therefore recommended to terminate the flow immediately.

3.1.5 Severe Congestion Handling Operation

Severe congestion is an error state in a node. It can occur because of route changes, a link failure or just a long period of congestion. When severe congestion occurs, the ingress node must be informed. If the severe congestion occurs in the interior node, then these nodes will set the severe congestion flag in the PHR signaling message and will forward it to the egress node. The egress node will inform the ingress node by sending a report message with the severe congestion flag set. After receiving this message, the ingress node will discard all new incoming requests for the severely congested path, for a certain time.

A flow diagram showing the severe congestion handling is depicted in Figure 3-5, where the severe congestion flag is set in a PHR_Refresh_Update and as a result, no data traffic is sent to that communication path. Note that this separation is only for illustrative purposes, since once a severe congestion occurs in the path independently of which messages are marked with severe congestion, all flows sent on that path will be blocked. Optionally, the ingress may notify the sender using the external protocol with a tear down message.

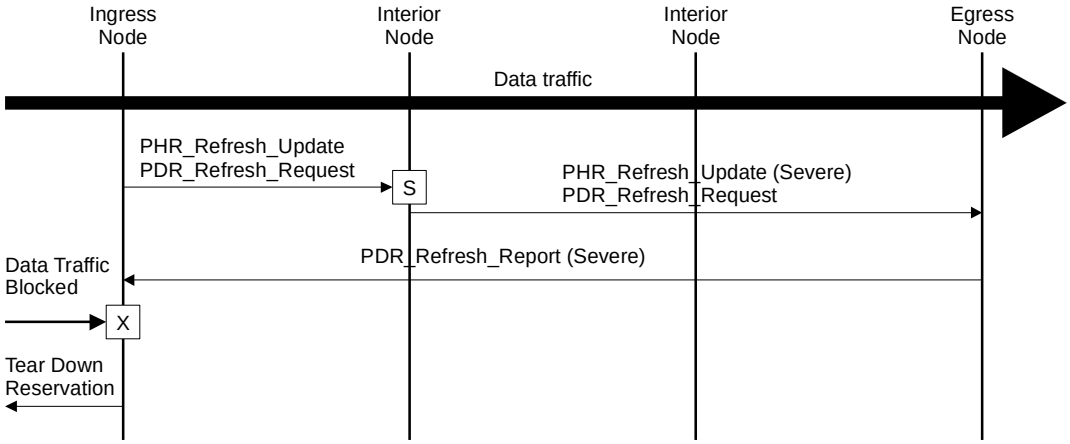

Figure 3-5: Severe congestion operation

3.2 RMD on Demand PHR

The RODA PHR is a reservation-based PHR protocol implementing the following functions:

- Management of one reservation-state per DSCP using soft state reservations. The reservation-state is a just number of reserved resources or units.
- Stores a pre-configured threshold value of the maximum allowable resource units per DSCP.
- Adaptation to load sharing. Load sharing allows interior nodes to take advantage of multiple routes to the same destination by sending via some or all of these available routes. The PHR protocol can handle load sharing if it is used.
- Severe congestion notification. A PHR message can notify the edges about the occurrence of a severe congestion situation.
- Transport of transparent PDR messages. The PHR protocol may encapsulate and transport PDR messages from an ingress node to an egress node.

RODA PHR specifies both the PHR_Resource_Request and the PHR_Refresh_Update messages. The QoS parameter consists of only one parameter, the number of resource units. One unit can represent a small slice of bandwidth or it may be defined using other more sophisticated methods, such as effective bandwidth.

The reserved resource units need to be refreshed regularly, so the ingress node will generate a PHR_Refresh_Update message in order to refresh the PHR aggregated soft state in both interior and egress nodes. The resources are released if there are no PHR_Refresh_Update messages received during a refresh period and this is the way reservations have to be released in the RODA PHR. It means a slight ineffectiveness, since the resources may be reserved up to one more refresh period after the flow was terminated. However, this trade-off is acceptable when no per-flow states are necessary in the core.

3.2.1 RODA PHR Message Format

The RODA PHR protocol messages are placed in an IP header options field, as defined in [RFC791], when IPv4 is used. [WeJa01b] also specifies how to use RODA PHR in an IPv6 packet [RFC2460]. The RODA PHR IPv4 option header field has the following format:

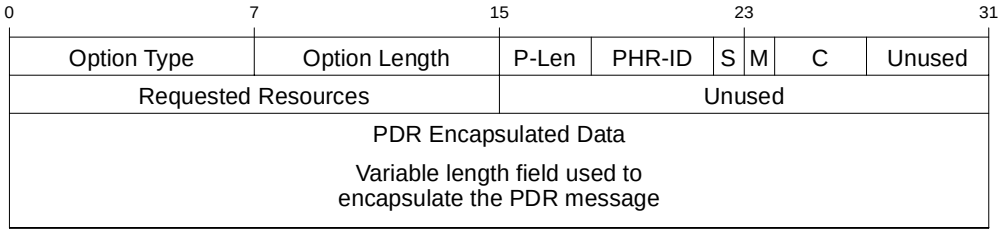

Figure 3-6: RODA PHR option field in the IPv4 option header

Option Type	Identifier of the type of option. The semantics of this field are specified in the IP standard document [RFC791]. A specific identifier for PHR messages must be specified.
Option Length	This field is specified in [RFC791] and represents the length of the data of this option, in octets
P-Len (PHR Length)	This field specifies the length in octets of the specific PHR information data. This information does not include the encapsulated PDR information. Zero specifies that this IP option field contains only PDR information and that it starts where the Requested Resources field starts.
PHR-ID	This field specifies the PHR type. For the RODA PHR this field must be set to one.
S (Severe Congestion)	This flag is set by an interior node when a severe congestion situation occurs. The ingress node sets this flag to zero.
M (Marked)	This flag is set by an interior node when the node cannot satisfy the reservation. The ingress node sets this flag to zero.
C (Message Type)	This field specifies the type of the PHR message. It can be either the value one for PHR_Resource_Request or the value two for PHR_Refresh_Update.
Requested Resources	This 16-bit field specifies the requested number of resource units to be reserved by all nodes in the path.
PDR encapsulated data	PDR encapsulated information data. This field is only processed by the edge nodes.

Note that the PHR message does not include the ingress node address, so when a PHR message reaches the egress, no report can be sent back. RODA is therefore dependent on the PDR protocol for this reason. A PHR message must be sent together with a PDR message and the PDR message must include the ingress address. The ingress address is not needed by the interior nodes and is therefore put into the PDR message.

3.3 The Per Domain Reservation Protocol

The PDR protocol works as a signaling protocol between the ingress and the egress nodes only. It is used to send important information, such as reservation states, between the two nodes. It also specifies how the service offered by the external protocol is going to be handled in the domain.

There may be several PDR protocols specified, either because different external protocols need to be supported, or because of other network requirements. A PDR may use either several different PHRs or only one. The RMD framework defines a common set of features that need to be realized by any PDR that uses a specific PHR, such as the RODA PHR. These features are described in the following sections. The PDR must also specify the service mapping and the message interaction with the supported external protocols. In general, a PDR protocol implements all, or a subset, of the following functions:

- Mapping of external QoS requests to DSCPs.
- Resource reservation within an entire Diffserv domain, including maintenance of flow identifiers and reservation states per flow.
- Notification of the ingress IP address to the egress node.
- Notification of lost signaling messages occurred in the communication path from the ingress node to the egress node.
- Notification of resource availability in all the nodes located in the communication path from the ingress node to the egress node.

Depending either on the external resource reservation scheme with which the Diffserv domain has to inter-operate or on the characteristics of the network, the RMD framework specifies that several PDRs can use one PHR. For example, a cellular network might use a different PDR for radio frame traffic than the PDR that is used for Intserv and RSVP in the same network. However, both PDR protocols may use the same PHR protocol. For a PDR to co-operate with all PHR protocols, it needs to implement a certain set of common features. Some of these features are described in the following sections.

3.3.1 Ingress Node Addressing

There are many situations, such as acknowledgement of a request, when the egress node has to notify the ingress node about the resource reservation status or something else. This means that the egress node must be able to send a PDR signaling message to the ingress node. Depending on the PDR used and the characteristics of the underlying network, the IP address of the ingress node can be derived in at least two ways:

- The egress node can determine the IP address of the ingress node from the available information contained in the header of a received PHR signaling message. For example, this could be the source IP address of the PHR signaling message received.
- The ingress node puts its IP address in the PDR signaling message, which is then sent together with a PHR signaling message. The egress node decapsulates the received PHR message and is able to extract the ingress IP address from the PDR message.

The first option is, in general, not possible, because the PHR and PDR messages need to follow the same route as the data packets for the flow they belong to. This means that the ingress cannot have its own IP address in the PHR signaling messages it generates. Therefore, the ingress address has to be introduced in the PDR message.

3.3.2 Error Control

The PHR signaling messages may be dropped in the communication path from the ingress to the egress nodes. If this occurs, the PDR protocol must be able to handle the recovery of the dropped reservation-based and measurement-based PHR signaling messages. This can be handled in different ways, such as stop the flow or continuing as if nothing happened. If the external protocol has a function to notify the sender that the flow is stopped, then it might be reasonable for the PDR to use this functionality. Otherwise, the PDR might try to cover up the situation by re-sending the reservation message. The choice depends also on the PHR protocol and its characteristics.

3.3.3 Management of Reservation States

The per-domain reservation functionality must support the initiation and maintenance of PDR states. This can be accomplished by using either a newly defined PDR protocol or one or more already existing protocols. Examples of such existing protocols are RSVP, RSVP Aggregation, SNMP and COPS. In some cases, the PDR protocol can be the same as the external protocol, but probably with some minor modifications.

The reservation states are associated with the flows defined by the external protocol. They must be identified using the same flow identification used by the external resource reservation protocol. The egress node must also be able to identify the flow using the same flow identification after receiving a PDR signaling message. Depending on the external protocol and the underlying network characteristics, the flow identification can be derived in at least three ways:

- Derived from already existing packet headers. The flow identifier can be derived from the available information contained in the IP header or the PHR signaling message. This could, for example, be the combination of the source and destination IP addresses and the DSCP in the IP header.
- Explicitly carried in the PDR message. The flow identifier is included in the PDR signaling message, which is sent together with the PHR signaling message. The egress can extract the flow identification from the PDR signaling message.
- A combination of above. The flow identifier can consists of some fields in the IP header and PHR message and also some extra information in the PDR message. For example, the flow identification can consist of the destination address and protocol number in the IP header together with a port number carried in the PDR message.

Moreover, the PDR report messages that are sent by the egress node back to the ingress node must also contain the flow identification. The ingress must know which flow a certain report refers to.

3.3.4 Resource Unavailability

When there are insufficient resources available in the communication path between the ingress node and egress node, the ingress node must be notified by means of a PDR report message. Any interior node that cannot admit a reservation request due to lack of available resources must be able to mark the PHR signaling message that will be sent toward the egress node. The egress node will then generate and send a marked PDR signaling message to the ingress node to indicate that an interior node could not admit the reservation request. Upon receiving this message, the ingress will reject the external resource request since there are insufficient resources available internally to satisfy this request. The PDR report message must support this.

3.3.5 Interoperability with External Resource Reservation Schemes

The RMD framework is initially designed for a single edge-to-edge Diffserv domain. However, as part of the global Internet, this single edge-to-edge Diffserv domain will have to inter-operate with other domains that may or may not be Diffserv-capable and which may use different resource reservation schemes. The RMD framework, which is specified as an open framework, must be able to inter-operate with these external resource reservation schemes. That is, the PDR functionality will have to take care of interoperability and service mappings between the external resource reservation schemes and the PHR protocols.

The ingress must map the external requests into appropriate PHBs and PHRs. This is called service mapping and consists of appropriate selection of PHB, admission control and policy control. This is based on the external QoS request, the available resources and other policies in the Diffserv domain. The policy may depend on SLAs or on the different types of services offered by the external resource reservation protocol. Both the PHB and the PHR must be able to handle the requested QoS and therefore the service mapping must be a part of the PDR protocol.

The PDR protocol must also define the interaction with the external protocol messages. The edge nodes must be able to process all the messages from the external protocol and must be able generate replies and reports on them. The PDR protocol must support all the features in the supported external protocols with only the help from Diffserv and RMD functionality.

3.3.6 PDR Message Format

The PDR message can be carried either inside the PHR IP option or somewhere else in the IP packet. The RODA PHR offers the opportunity to encapsulate a PDR message in its IP option. However, that space is limited and it is sometimes necessary to place the PDR message somewhere else. In this section, an example PDR message is discussed. Figure 3-7 shows an example of a PDR message format.

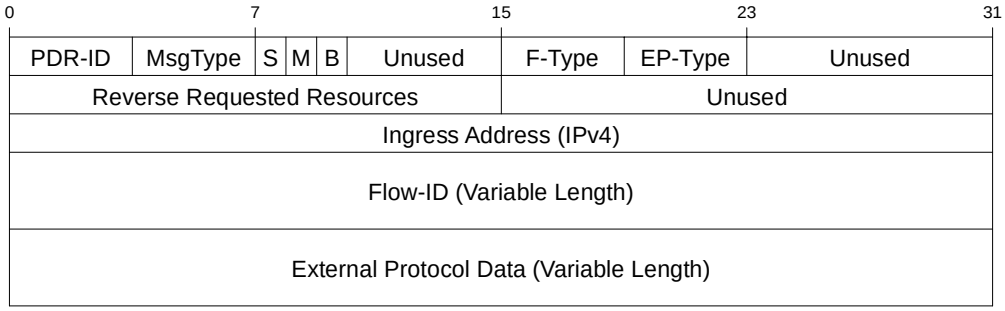

Figure 3-7: PDR message structure for IPv4

PDR-ID	A value that identifies the PDR protocol.
MsgType	The type of PDR message. Section 3.1.2 describes the possible message types.
S (Severe Congestion)	The severe congestion flag was set in the PHR_Reserve_Request or PHR_Refresh_Request message. This flag only applies to PDR report messages.

M (Marked)	This flag is set if the marked flag was set in the corresponding PHR_Reserve_Request or PHR_Refresh_Request message. This flag only applies to PDR report messages.
B (Bi-directional)	The request is Bi-directional. This flag only applies to PDR request messages. The egress node will make a reservation in the opposite direction using another PHR message when it receives a PDR message with B set.
F-Type (Flow Type)	The Flow-ID type identifier. Defined by the PDR protocol. It tells the ingress and egress nodes what kind of data is contained in the Flow-ID and its length. Every edge node must be configured to understand all used F-Types.
EP-Type (External Protocol Data Type)	Identifies the external protocol data. It tells the ingress and egress nodes what kind of data is contained in External Protocol Data field and its length. Every edge node must be configured to understand all used EP-Types.
Reverse Requested Resources	This 16-bit field specifies the requested number of units of resources that have to be reserved by a router in the reverse direction when bi-directional reservation functionality is supported.
Ingress Address	This is the ingress address when the PDR message goes from the ingress towards the egress. This field is not used for report messages in the direction towards the ingress.
Flow-ID (Flow Identifier)	This part is dependent on the F-Type. For example, this may be the normal RSVP flow identifier consisting of destination IP address, protocol ID and destination port. The length would then be 7 bytes in the IPv4 case. Sometimes it is only necessary to use this field to match reports with requests. In that case, it is enough if the Flow-ID is a 16-bit number.
External Protocol Data	This field is used when tunneling external protocol data between the ingress and egress. Any kind of data may be placed here, but it must be specified by the PDR for each valid EP-Type. When using RSVP, this may be the token bucket specifier used by RSVP.

3.4 RMD in IP-based UTRAN

The RMD scheme is designed to be applicable to any type of IP-based core network and any type of IP-based access network. The network can be both wired and wireless, as long as it is using the Diffserv edge-to-edge architecture. The specific characteristics of the UTRAN constrain the resource management strategies applied in an IP-based network with strict requirements. As stated in chapter 2, these requirements are not met by the current resource management schemes. The RMD on the other hand, satisfies these specific resource management requirements and gives the RMD framework an advantage over the current resource management strategies in IP-based UTRAN.

The transport network in the UTRAN is used by only one single operator or a few trusted ones. This means that one single Diffserv domain can be used in the transport network. Furthermore, no policing or shaping because of security reasons is necessary in the edges. All the nodes in the domain can be trusted. Figure 3-8 shows how the Diffserv domain can look like in an UTRAN environment. In Figure 3-8, the RNCs and Node Bs are connected to the Diffserv domain via edge nodes. It is also possible that the RNCs and Node Bs are physically the same equipment as the edge nodes.

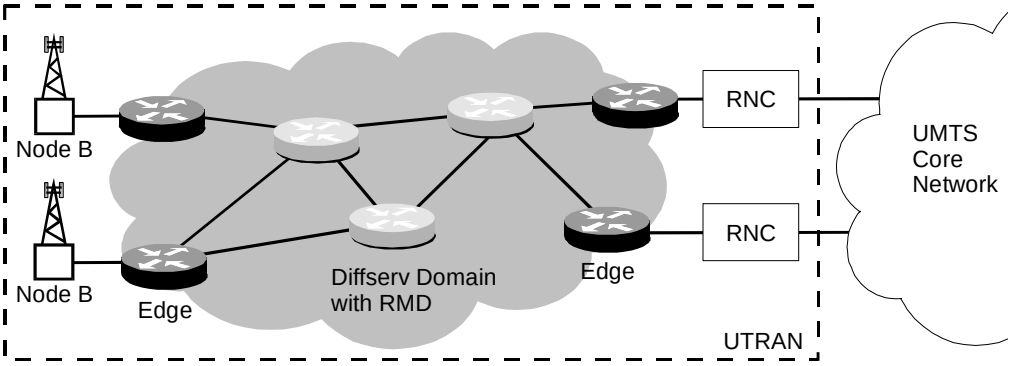

Figure 3-8: Diffserv domain in IP-based UTRAN

If the RNC and Node B are not the edge nodes, as in the figure above, it is necessary to have an external resource reservation protocol between them. This protocol must support the requirements from UTRAN and must be able to reserve resources for the radio frame flows. It is a sender-initiated quantitative reservation protocol, which should be able to inter-operate with the QoS mechanisms used in the mobile stations and with the IP QoS architecture used in the core or network. If the Node Bs and RNCs are the edges, this protocol is not a message protocol in the usual meaning. Instead, it specifies how the applications in the node can communicate their resource requirements to the RMD functionality in the same node. This can, for instance, be implemented using system calls.

There is also a need for a UTRAN specific PDR protocol. The PDR must support all the features described in section 3.3 and also support bi-directional reservations in co-operation with the external protocol. The bi-directional feature can be accomplished by the UTRAN PDR by combining two uni-directional reservations. It has to be solved that way, because packets traveling from the ingress node that initiates the reservation towards the egress node are likely to follow a different path than packets traveling in the opposite direction. The bi-directional reservation imposes two requirements on the edge nodes as described below:

- The edge nodes must be able to distinguish between a uni-directional and a bi-directional resource reservation PDR message. This can be accomplished by using a flag in the header of the PDR signaling messages. Furthermore, these bi-directional packets must include the requested resource parameters for initiating a uni-directional reservation in the reverse direction. Note that the requested resource parameters used for bi-directional reservations are asymmetric, the values of the requested resources can be different in the two directions.

- When an egress node receives a bi-directional PDR reservation request message, the egress node will have to construct an uni-directional PDR resource reservation message and a PHR message, which will be sent in the opposite direction.

This thesis describes only the RODA PHR, which is an obvious alternative for managing the radio frame traffic resources in UTRAN. However, it might be possible the use a measurement-based PHR protocol instead. A measurement-based protocol would assume that the flows do not change their traffic characteristics too much. Since this is the case, such a PHR protocol might also be a good alternative.

[4] PROTOTYPE IMPLEMENTATION

In this chapter, a prototype implementation of RMD in IP-based networks, such as the IP-based UTRAN, is described. In the prototype, it is assumed that the sender and receiver hosts are not part of the Diffserv domain and are located outside of this domain. Furthermore, a simple resource reservation protocol was developed that can be used on an end-to-end basis. This protocol is called Simple External Protocol (SEP) and it is designed to be as simple as possible to implement.

Moreover, a PDR protocol that can inter-operate with the SEP protocol is also defined. This PDR, called Simple PDR, can be used with the RODA PHR, which is the only PHR implemented in this prototype. The main purpose of the first prototype is to explore RMD, test different RMD features and compare it with other resource management schemes.

This chapter starts with defining the SEP protocol and the Simple PDR. Then, the Linux environment is described, with the traffic control parts of Linux emphasized. Finally, the design of the prototype is presented as it was implemented.

4.1 Simple External Protocol

The main goal of the SEP prototype is to test the RMD functionality. SEP is designed to be as simple as possible and it fulfills the requirements for transmitting radio frames in an IP-based UTRAN. Only a few requirements are not offered by the SEP protocol, such as the bi-directional reservation feature.

The SEP protocol uses resource reservation states that are managed by explicit reserve and release request messages, without using the soft-state principle. By not using the soft-state principle, the management of the reservation states will require explicit termination of the reservations. However, this principle makes the implementation of the SEP protocol much simpler, since the refresh process of the reservation states is not needed.

Furthermore, SEP is a sender-initiated, reservation-based, uni-directional protocol and used on an end-to-end basis. Every SEP-aware node located between the sender and the receiver needs to reserve resources and schedule packets per flow. The SEP does not comply with Intserv, but it still uses quantitative reservations with a simplified token bucket specifier (see Section 2.1.2). The main QoS service offered by SEP to the reserved flows is assurance of few packet drops. No delay guarantees are assured by the SEP protocol.

Due to its implementation simplicity, SEP lacks some important features. In particular, SEP does not support load sharing, route changes, multicast reservations and IPv6 [RFC2460].

4.1.1 SEP Message Format

SEP uses the following message types:

- Reserve. This message is sent from the sender towards the receiver and it is used to initiate the reservation state in each SEP-aware node located on the communication path. Every SEP-aware intermediate node must process this message.
- Release. The release message is sent from the sender towards the receiver and it is used to release already initiated reservation states in each SEP-aware node located on the communication path. Every SEP-aware intermediate node must process this message.
- Reservation Accepted. This message is sent from the receiver to the sender to notify that the reserve request has been accepted.
- Reservation Rejected. This message is sent from the receiver or any intermediate node to the sender to notify that the reservation was rejected.
- Tear Down Flow. This message can be sent by any intermediate node or the receiver to the sender to initiate the release of a reserved flow. The sender must then immediately release the flow by sending a release message towards the receiver.

The SEP messages are encapsulated in IP datagrams. Figure 4-1 shows the structure for all SEP messages. The SEP flow identifier consists of the source IP address, the protocol and the source port. The source IP address is taken from the IP header of the SEP packet and is therefore not in the SEP message itself. The SEP protocol is encapsulated in the payload of the IP packet using a specific protocol number. In this prototype, number 63 was used. That number is assigned by IANA for internal use.

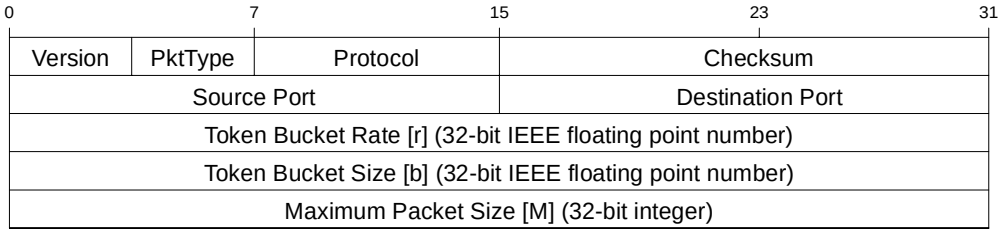

Figure 4-1: The signal packet structure

Version	The SEP version used. Must be set to zero.
PktType	The message type identifier and can be one of the messages mentioned above.
Source Port	The source UDP or TCP port for the data flow. This field is part of the flow identifier and is used for flow classification in the SEP nodes.
Destination Port	Some load sharing implementations use a hash function to determine which outgoing interface to forward the packet to. Those hash functions typically use source and destination addresses and ports as input. As will be shown later, this field is used by the RMD domain to make sure signaling messages and data packets follow the same path.

Protocol	The protocol used in the data flow. Must be either TCP(6) or UDP(17). This field is part of the flow identifier and is used for flow classification in the SEP nodes.
Token Bucket	The last 12 bytes of the message are a token bucket specifier. It is a stripped version of the token bucket used in RSVP, see section 2.1.2. Peak rate and minimum policed unit are removed. Everything is in bytes or bytes per second.

4.1.2 SEP Operations

When a sender host wants to reserve a flow to another host, it generates a SEP Reserve packet and forwards it in the same direction as the data traffic for the flow. The packet has the IP router alert option [RFC2113], so that every intermediate SEP-aware node knows that it has to process it. Every node along the forwarding path has to reserve the requested resources or reject the reservation. When the SEP Reserve packet reaches the receiver, the receiver sends back a SEP Reservation Accepted packet to the sender to inform about the successful reservation. The whole process is shown in Figure 4-2. On the other hand, if a node rejects the reservation, it must send a SEP Reservation Rejected packet directly back to the sender. The IP Router Alert Option is not needed for packets going to the sender, such as the SEP Reservation Accepted and SEP Reservation Rejected packets, since only the sender needs to process the packet. Only the SEP Reserve and SEP Release packets need to be processed by each intermediate nodes. Therefore, these packets will have to include, and use, the IP Router Alert Option.

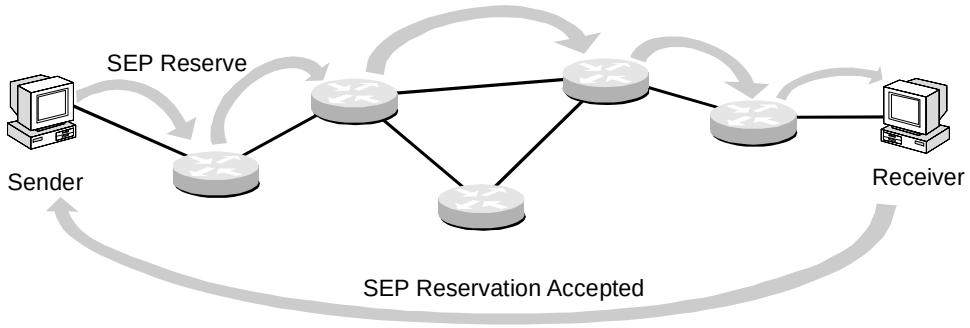

Figure 4-2: A successful SEP reservation

When the sender decides to terminate the flow, it sends a SEP Release packet in the same forwarding path. Every intermediate node releases the reserved resources for the flow and forwards the SEP Release packet to the next hop. Only the sender can terminate a flow, but any intermediate node or the receiving node may send a SEP Tear Down Flow packet to the sender. The sender must then immediately release the flow by sending a SEP Release packet. The sender must also send a SEP Release packet if it receives a SEP Reservation Rejected packet. The reason of this is that some nodes in the beginning of the communication path may have reserved the resources and therefore, they need to be released explicitly.

4.2 Simple PDR

In this prototype, the Diffserv domain must use a PDR protocol that will be capable of offering the QoS service requested by the SEP protocol. This PDR, called Simple PDR, uses the RODA PHR to manage the resources inside the domain. The Simple PDR is only an example of how the SEP service can be offered by an RMD-enabled Diffserv domain. It is based on the example PDR discussed in section 3.3. However, there are other alternatives to the Simple PDR, such as using a modified SEP protocol.

4.2.1 Service Mappings

The service offered by the Simple External Protocol does not assure any delay or delay variation guarantee. However, the SEP protocol assures few packet drops and therefore, a mapping for the reserved traffic into an EF [RFC2598] class is possible. The network operator can have several EF traffic classes and may decide to classify the reserved traffic into different EF classes based on the sender address, receiver address or based on some other kind of policy. The EF classes can, for instance, have static assigned bandwidth resources. By mapping different senders into different EF classes, the network operator can decide which senders share the resources and how much they may reserve. In this case, each EF class has an upper resource limit.

The RODA PHR uses the concept of resource units. The PDR must define exactly what a unit is, since this depends on the offered service. In the Simple PDR, a unit is a slice of bandwidth. Each EF class has a limit of how much bandwidth it may utilize. This upper limit is mapped into a number of available units by dividing the upper limit bandwidth with the bandwidth represented by a unit. If an EF class is assigned 2 Mbps and a unit is defined to be 4 kbps, then there are 500 units available for reservations in that traffic class.

When a reservation is made with SEP, the reservation is specified with a token bucket specifier with bucket rate, bucket size and maximum packet size. This is mapped into units, using only the token bucket rate, by the ingress node. However, the ingress node may also use other parameters from the token bucket specifier when admitting new flows locally.

There is no recommended bandwidth value for a unit, but there are a couple of things to take into consideration when defining a unit. If one unit is 4 kbps, then it is not possible to make reservations for flows with a bandwidth greater than 4 * 65535 = 262 Mbps. If that is necessary, a unit must be equal to more bandwidth or the ingress node must send several PHR reservation messages for the same flow. Furthermore, the unit bandwidth should also be a divider of the most common reservations. If reservations of 5 kbps are common, then it is necessary to reserve 2 units for these flows. This means that we reserve 3 kbps too much per flow, which are not utilized. In this case, a unit of 1 kbps or 5 kbps is preferable.

4.2.2 Simple PDR Message Format

The Simple PDR message format is the same as the PDR described in section 3.3.6 and implements all features discussed in section 3.3, except the bi-directional flag. The following messages are defined by the Simple PDR:

PDR_Reservation_Request	This message is sent from the ingress node, together with a PHR_Resource_Request, when a SEP reserve message is received. It must carry both the token bucket specifier and the flow identifier so that the egress node can re-construct the original SEP reserve packet.

PDR_Refresh_Request	This message is sent to refresh the PDR state in the egress node. It must contain the flow identifier so that the egress can associate the message with the correct flow.
PDR_Release_Request	This message is sent by the ingress when a SEP release message is received. It will release the SEP and PDR states in the egress node. It must carry the flow identifier, so that the egress node can re-construct a SEP release packet.
PDR_Reservation_Report	This message is sent by the egress node on reception of a PDR_Reservation_Request. It must contain the same flow identifier as the request message, so that the ingress can match the report with the correct flow.
PDR_Refresh_Report	This message is sent by the egress node on reception of a PDR_Refresh_Request. It must contain the same flow identifier as its request message.
PDR_Release_Report	This message is sent by the egress node on reception of a PDR_Release_Request. It must also contain the same flow identifier as its request message.

The flow identifier used in the PDR messages is always the same as the one used in SEP and it looks like in Figure 4-3, when sent in a PDR message. This flow identifier has F-Type number one in the PDR message header and must always be sent in a Simple PDR message.

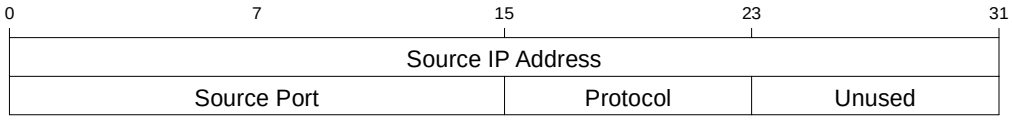

Figure 4-3: Flow identifier in Simple PDR

In the PDR_Reservation_Request and PDR_Refresh_Request, the following external data is necessary. In all the other packets, no data is needed. EP-Type one in the PDR message header is associated with this external protocol data, while value zero means no external protocol data.

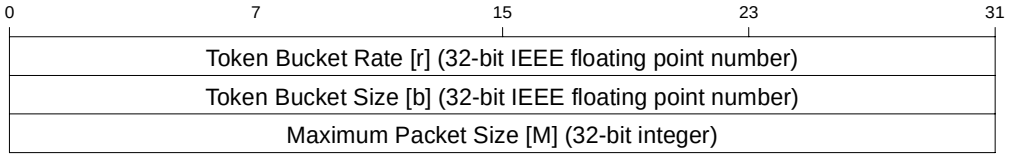

Figure 4-4: External Protocol Data field in some Simple PDR messages

Note that the length of the PDR message plus the flow identifier and the external protocol data is 32 bytes. This is exactly what can fit into the PHR message, since the PHR header is 8 bytes and 40 bytes is the maximum length of the IP options [RFC791]. If a more complicated

token bucket specifier is required, than some part of the PDR message will have to be placed somewhere else.

4.2.3 Message Interaction with SEP

Assume that a sender wants to reserve resources for a flow, which has to cross an RMD-enabled Diffserv domain and therefore sends a SEP Reserve packet towards the receiver, as shown in Figure 4-5. First, the SEP Reserve packet reaches the ingress node of the domain and then the ingress performs a local admission control based on the SEP mechanisms and the token bucket specifier in the request. It maps the request into a DSCP and translates the token bucket specifier into units for the RODA PHR. Then it sends a PDR_Reservation_Request message encapsulated in a RODA PHR_Resource_Request message towards the receiver. The message is sent in an IP packet with the PHR and PDR messages in the IP option. The payload consists of a dummy UDP or TCP header containing the correct source and destination port numbers. This is because load sharing based on these fields may be used inside the domain. Even if SEP does not support load sharing and route changes, RMD does.

When the egress node receives the packet, it will first check the M- and S-flag in the PHR_Resource_Request message. If the reservation was successful, it will extract the flow identifier and the token bucket specifier from the PDR message and send a SEP Reserve packet to the next hop. At the same time, it will send a PDR-only message containing a PDR_Reservation_Report message with the same flow identifier to the ingress node. On the other hand, if the reservation was not successful, no SEP message will be generated and the S- and/or M-flag in the PDR report message will be set in accordance with the received PHR_Resource_Request message.

When the ingress receives the PDR report, it will update the packet scheduling with the new flow. The ingress will perform policing and shaping on the new flow as specified by the SEP request. Even the egress node may perform per-flow scheduling with shaping if necessary. However, only the ingress and egress nodes will perform SEP functionality, the interior nodes use only the RODA PHR functionality. The ingress node must understand the SEP Reserve and SEP Release messages and must be able to send the SEP Reservation Rejected and SEP Tear Down Flow messages. The egress node must be able to send a SEP Reserve message to the next hop.

Finally, the SEP request reaches the receiver, which will send back a SEP Reservation Accepted message to the sender. The sender may, at this point, start to send traffic. Figure 4-5 demonstrates the entire process of a successful reservation using SEP in a Diffserv domain with RMD and the Simple PDR

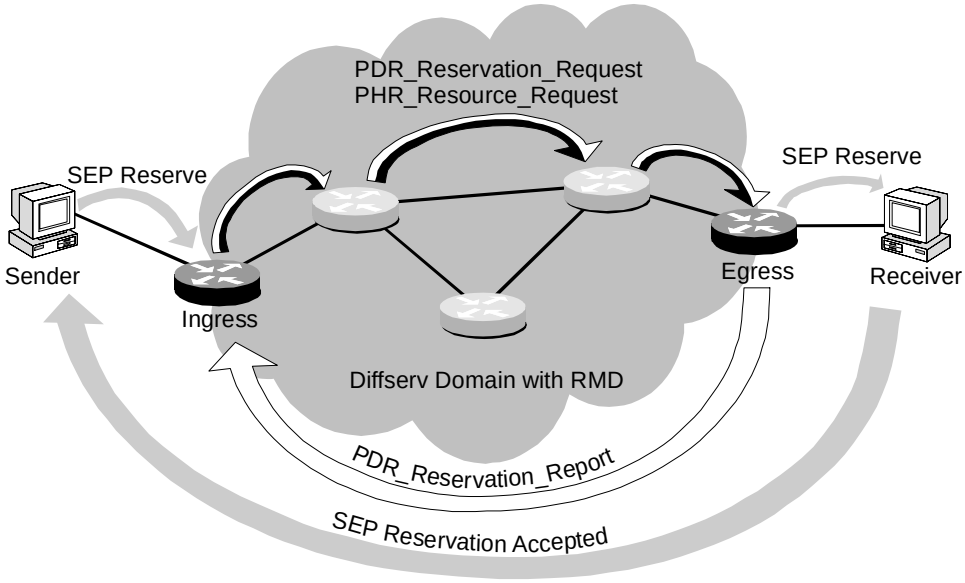

Figure 4-5: SEP together with RMD

The ingress node is responsible for the flow in the entire Diffserv domain. If it experiences a timeout or a marked report message, it may stop the flow and send a SEP Tear Down Flow message to the sender. There may be other reasons why an ingress node wants to terminate a flow. However, it should send a SEP Tear Down Flow message and wait for the sender to release the flow with a SEP Release message. The ingress node must not remove the flow state until that message arrives. This is not a problem since SEP does not support re-routing and will never change ingress or egress nodes. However, the communication path between the ingress and the egress can change.

4.3 The Linux Implementation Environment

Linux is an open source operating system, which has been in development since 1991. It is a complete UNIX-like operating system with support for a number of advanced networking features. There are good developing tools available, like compilers, editors, debugger and more. For these resaons, Linux is a good candidate for prototyping network protocols, such as the RMD.

The Linux kernel has extensive support for Quality of Service networking, including a flexible framework for packet classification and scheduling [Alme99], [Radh99]. The Linux QoS architecture has been developed over several years and now includes support for both Intserv and Diffserv. Figure 4-6 briefly shows the network design in Linux. This figure shows how the kernel processes incoming packets and how packets from higher layers are processed. The input de-multiplexer examines the incoming packets to determine if the packets are destined for the local host. If so, they are sent to the higher layers, or if not, they are sent to the forwarding block. The forwarding block finds the correct outgoing interface for each packet using a routing table. The packet is enqueued on a queue associated with the interface before transmitted on the physical line for the interface.

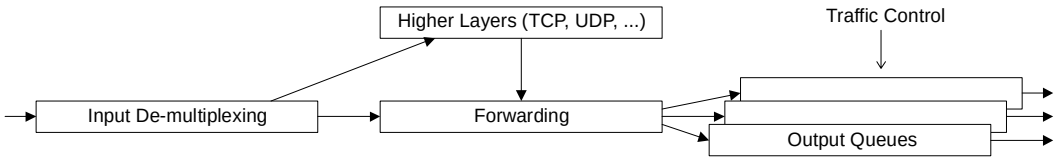

Figure 4-6: Conceptual overview of networking in Linux

It is in the output queue of the interface where the Linux traffic control comes into play. The traffic control can decide which packet to send first, if packets should be dropped, to schedule packets for later transmission and more. The default output queue is an ordinary First In First Out (FIFO) queue with tail drop. Nevertheless, this can be changed by a configuration tool called tc, available in the iproute2 package [Radh99]. With tc, a queuing structure consisting of policers, meters, classifiers, droppers and schedulers (see [RFC2475]), can be set up for each outgoing interface.

4.3.1 The Queuing Structure

The queuing structure consists of the following two types of components:

- Queuing disciplines, such as schedulers, shapers and queues.
- Filters, such as packet classifiers and policers.

A queuing discipline may have something called classes. Classes can be used by a queuing discipline to divide the traffic into different traffic classes. A queuing discipline without classes is just a single queue. An example is the normal FIFO queue, which drops packets at the tail when it is full. Another queuing discipline example is Random Early Drop (RED) [FlJa93]. That queuing discipline is also a FIFO queue, but it uses the RED method instead of tail drop. A queue using RED starts to drop packets even before it is completely full. The probability of dropping packets increases as the queue gets more full. A third classless queuing discipline example is the token bucket shaper. It is also a FIFO queue, but it performs shaping by delaying packets if necessary.

Figure 4-7 shows a queuing discipline that has two classes. A queuing discipline with classes has a list of filters that filter enqueued packets into one of the classes. There are different kinds of filters, but most of them filter on packet header fields. In Linux, a filter can also contain a traffic monitor and a policer. When a packet is assigned a class, it is usually enqueued on the queuing discipline associated with that class. That queuing discipline can be any type of queuing discipline, both a single queue or another queuing discipline with classes.

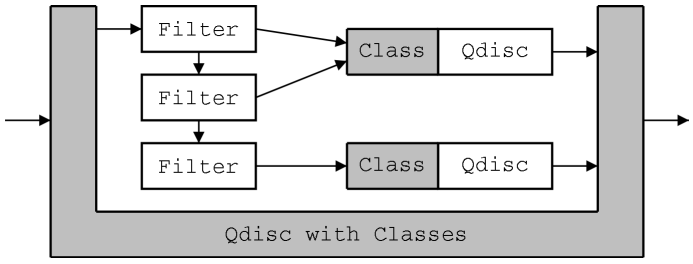

Figure 4-7: Queuing discipline with two classes and three filters

When the network interface is ready to transmit the next packet, the interface dequeues one packet from the queuing structure and sends it. In the case of a classful queuing discipline, the queuing discipline will choose a class from where a packet will be dequeued. This process is called scheduling and there are a number of different ways of doing this. For example, the priority queuing discipline assigns priorities to the classes and will dequeue a packet from the class with the highest priority first. If that queue is empty or cannot transmit a packet because of shaping, the next class is examined until a packet for transmitting is found. If the chosen class has a classful queuing discipline, the scheduling algorithm for that queuing discipline is activated. It will choose one of it classes and the procedure will repeat until a packet is dequeued from a real packet queue.

The Class-based Queuing (CBQ) scheduling algorithm is described in [FlJa95]. Linux implements this algorithm in the CBQ queuing discipline, which is an important example of a classful queuing discipline and is heavily used in Diffserv. The CBQ queuing discipline divides different traffic flows into different priority classes. Each class is assigned a fixed slice of bandwidth and organized in a tree structure. However, it is possible to configure CBQ classes to borrow bandwidth from other classes. In Linux, the CBQ queuing discipline uses a weighted round robin (WRR) scheduling algorithm for implementing CBQ.

4.3.2 Diffserv Using the Linux Traffic Control

Diffserv is implemented in Linux, by extending the traffic control with some new queuing disciplines and packet filters [AlHa99]. A new queuing discipline, called dsmark, is implemented to retrieve and change the DSCP value of IP packets. It can be used to perform marking and re-marking of IP packets. A filter, called tcindex, implements filtering based on the retrieved DSCP value and can be used to classify packets into their correct PHB traffic classes. A generalized RED queue (GRED) is implemented to support the AF PHB [RFC2597]. The traffic class scheduling can usually be implemented using already existing queuing disciplines, such as the priority and CBQ queuing disciplines.

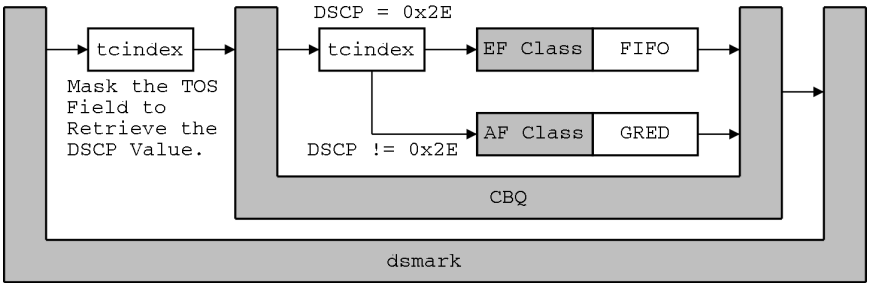

Figure 4-8: Diffserv interior node traffic scheduling example in Linux traffic control

Figure 4-8 shows an example of the traffic control structure on an outgoing interface of a Diffserv interior node. The surrounding dsmark queuing discipline will retrieve the Type of Service (TOS) value from the IP header and save it in a variable, called tc_index. The first tcindex filter will mask that value and shift it, so that only the DSCP value is stored in tc_index. Note that the DSCP value is the six most significant bits of the TOS field. Then, the second tcindex filter will filter the packets depending on the value in the tc_index variable. If the value is 0x2E, the packet will be put in the FIFO queue for EF traffic, otherwise it will be placed in the GRED queue for AF traffic.

4.3.3 IP Router Alert Option

The goal of the IP router alert option [RFC2113] is to provide a mechanism whereby routers can intercept packets not addressed to them directly. The option has the semantic meaning "intermediate routers should examine the payload of this packet more carefully". By including this option in the IP header of the packet, a protocol can place its data in the payload instead of using IP options. This option was standardized because of RSVP, where all signaling messages must be processed by all intermediate routers and cannot be placed in the IP header as an IP option. SEP uses the IP router alert option for the SEP Reserve and SEP Release packets.

Linux has functionality for further examining packets with the IP router alert option before forwarding. A daemon that wishes to process all such packets using a specific protocol number, can open a raw IP socket and inform the system by calling the `setsockopt()` system call to deliver them to the daemon. The following example shows how to configure a socket to do this.

```
int s;
int true = 1;

s = socket(AF_INET, SOCK_RAW, 63);
setsockopt(s, IPPROTO_IP, IP_ROUTER_ALERT, &true, sizeof(true));
```

The input de-multiplexer in Figure 4-6 will now start to look for IP packets with the IP Router Alert Option and protocol number 63. If such a packet is found, and even if the destination address is another host, it will deliver the packet to the daemon and not forward it to the next hop. The daemon retrieves the packet with the normal `recvfrom()` system call, processes it and forwards it to the next hop by calling the `sendto()` system call. All the system calls described here are defined by POSIX and are further explained in [Stev98]. However, the IP_ROUTER_ALERT socket option is not yet part of any standard.

4.4 RMD Implementation

The Simple PDR, using SEP as the external protocol and the RODA PHR, were implemented in Linux version 2.4. The protocols were implemented as they are described earlier in this chapter with a few limitations. First of all, only flows using UDP can be reserved. Flows using TCP are not supported, but can easily be implemented in the future. Moreover, no real SEP admission control was implemented. The only implemented admission control is the one offered via RMD and the RODA PHR protocol. However, the goal of this prototype is to develop and study the RODA PHR and RMD framework, not the SEP protocol.

The Diffserv implementation in the prototype also has limitations. It has only two traffic classes, a best effort class and a premium traffic class using the EF PHB. All reserved flows are placed in the premium class. Both traffic classes have fixed and static bandwidth assignments without any resource borrowing between them.

Despite these limitations, it was possible to test all RODA PHR features and most PDR features. A fully functional implementation of RMD was possible. UDP flows can be reserved through a Diffserv domain with good resource management using RMD and the RODA PHR.

This section describes the design of the prototype by introducing all the components both inside the kernel and in user space. After that, the design of each type of node is described

and finally the all-round node implementation is described. However, first a sliding window algorithm used to maintain the PHR reservation states is described.

4.4.1 The Sliding Window Algorithm

All RODA PHR-aware nodes need to maintain the reservation states per traffic class and in case of no free resources, they will mark the PHR request messages. In the prototype implementation, a sliding window algorithm was used to maintain the reservation state for one traffic class and to decide when to mark a PHR packet. The sliding window algorithm divides the period into a number of cells and defines a window to be that number of cells. This means that a window has the same length as the refresh period used by RODA PHR. Now, assume that we have a reservation state like that shown in Figure 4-9. The current window is cell one to cell five and has 19 reserved units.

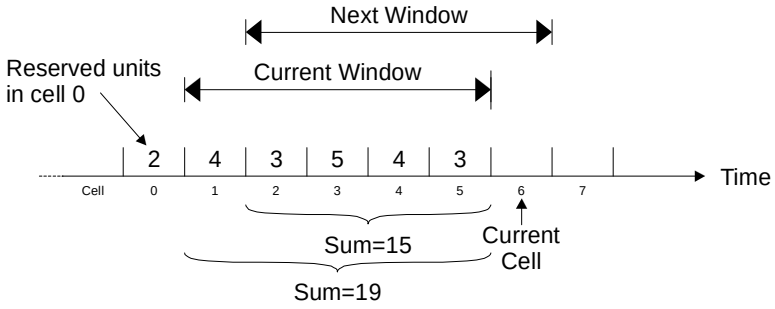

Figure 4-9: Sliding window

In cell number one, 4 units were reserved and since the current cell is cell six, one refresh period has elapsed since cell one. One refresh period is namely one window, which is five cells in the example above. This means, that if nothing happens to the flow, or flows, which reserved these 4 resources, they will be refreshed during the current cell. Therefore, the node expects to get refreshes for up to 4 units during this cell. However, a node may also receive refresh messages for flows that were not reserved before by the node. This occurs due to re-routing and therefore it is a good idea to admit these refresh messages if possible. The important thing is that the next window, beginning in cell two and ending in cell six, must not exceed the threshold. If the threshold is 20, the node may admit up to 5 units during cell six, since 5 + 15 is equal to the threshold.

When the node changes the current cell to cell six, a counter, called the refresh counter, is reset. Now, assume that a PHR_Refresh_Update message with 3 units arrives. Since 3 + 15 is less than 20, the refresh is admitted and the refresh counter is increased to 3. If another reservation is refreshed after that, requesting 4 more units, it must be marked, since 4 + 15 + the refresh counter (3) is 22, which is more than the threshold.

For new reservations, not only the sum of the next window must not exceed the threshold, but also the sum of the current window plus the reservation. Assume again that the refresh counter is zero in the example above, and a message requesting X resources for a new flow is being processed. This flow must be marked if X + 19 is more than the threshold, which is usually a stronger requirement than for refresh messages. The sum of the next window is less than the sum of the current window in the usual situation, since the refresh counter is less than the reservations in the first cell in the current window. However, this may not be valid during re-routing.

If the new reservation request is admitted, both the refresh counter and the first cell is updated. The first cell in the example will be X + 4, instead of 4, and the total sum for the current window will be X + 19. The pseudo code for the sliding window implementation is shown here:

```
nrofcells = 5               /* The window size in number of cells
*/
window[0..nrofcells-1] = 0  /* The reserved units in each cell */
rfcount = 0                 /* The refresh counter */
threshold = 20              /* The unit threshold */
first = 0                   /* Pointer the first cell in the
window
                               array */

On next cell:
    window[first] = rfcount
    first = first + 1 (modulo nrofcells)
    /* Note that the old rfcount now is in the 'last' cell in the
       window here */
    rfcount = 0

On PHR_Refresh_Update of X units:
    if X + sum(window) - window[first] + rfcount > threshold:
        reject
    rfcount = refcount + X
    accept

On PHR_Resource_Request of X units:
    if X + sum(window) - window[first] + rfcount > threshold:
        reject
    if X + sum(window) > threshold:
        reject
    rfcount = rfcount + X
    window[first] = window[first] + X
    accept
```

The overhead introduced by the sliding window algorithm is minimal. After a flow is terminated and it stops to send PHR refresh messages, the resources will be released within the maximum time of one refresh period plus one cell. A cell can be a very short time slot, since the extra processing time needed by the interior node is small. Note that the sum of the array in the pseudo code does not need to be calculated by adding all the values in the array every time. The sum can be stored in a single variable that is updated when a value in the window array is changed. Therefore, when more cells per window are used, it only means that the next cell event will be executed more frequently. The processing time of each event remains constant.

4.4.2 RMD Implementation Components

The RMD implementation consists of some existing kernel modules executed inside the Linux kernel and some new kernel modules. Some few modifications to the kernel code itself were necessary. Then, the prototype also has three daemons running as ordinary processes in the user space. The two daemons, called ingress daemon and egress daemon, were written in Python [Lutz96] and communicate with the kernel modules using the rtnetlink interface. The rtnetlink interface is an extension to the netlink socket [DhSu99] and looks like an ordinary

socket to the program. A program can configure the kernel and retrieve statistics and other information from the kernel via this socket. The third daemon, called interior daemon, was not implemented.

The kernel parts consist of four new queuing disciplines and three new filters. All can be compiled as kernel modules or permanently included in the kernel. The tc tool was modified, so that it can be used to setup and configure the new queuing disciplines and filters in the kernel. Below, all these queuing disciplines, filters and the daemons are briefly described.

4.4.2.1 The SEP Filter

This is a packet filter for the Linux traffic control in the kernel. It filters packets based on the SEP flow identifier, using a map from flow identifiers to classes. When a packet is being processed, the filter searches through a list of flow identifiers for a match. If a match is found, it enqueus the packet on the queuing discipline associated with the class. The SEP filter contains a policer for each flow and can be configured to discard packets not conforming to the reservations, if necessary. It is possible to dynamically add and remove flow identifiers and their classes via the rtnetlink interface or the tc tool and this is what the daemons in this RMD implementation do.

4.4.2.2 The PHR Filter

This is where the reservation state of the RODA PHR is located. This Linux traffic control filter only matches packets with the RODA PHR IP option. The RODA PHR reservation state is stored in this filter and updated as soon as a PHR message is filtered by this module. If the reservation cannot be made, it will mark the packet before it enqueus it on the queuing discipline for PHR signaling packets. The RODA PHR reservation state updating algorithm implemented in the PHR filter is described in section 4.4.1.

4.4.2.3 The Device Filter

This is a filter for the Linux traffic control and it filters packets based on their arriving interface. The filter has a list of interfaces and only packets that arrived on one of these interfaces match the filter. This filter is used by an edge node to find out if the packet is entering the domain or not. If a packet is entering the domain, it must first be classified and policed by the SEP functionality, otherwise it will be handled by the interior functionality only.

4.4.2.4 The PDRingress Queuing Discipline

The PDRingress queuing discipline is responsible for sending PHR and PDR reservation and refresh messages. It has only one class, so filters do not apply to this queuing discipline. The class can have any queuing discipline, but it is intended for shaping. When created in the kernel, it will send a PDR_Reservation_Request and PHR_Resource_Request packet towards the egress node. It will get the PDR report message when it comes back to the ingress node, through a small hack in the `ip_local_deliver()` function in the IP implementation of the kernel. If the report is unmarked, it will let the data traffic pass through and start a timer for sending refresh messages. The queuing discipline sends exceptional events via the rtnetlink interface to the ingress daemon about flow state changes. It will inform about marked PDR reports and timeouts, but as long as the refreshes are successfully retrieved, the flow will continue. In this implementation, the ingress daemon will listen for these events.

4.4.2.5 The PDRegress Queuing Discipline

The PDRegress queuing discipline is executed inside the kernel of the egress node and replies to the PHR and PDR signaling messages for a flow. At the same time, it discards them so that no signaling messages leave the Diffserv domain. The queuing discipline has a timer, and if it is not refreshed properly, it will inform a daemon by the rtnetlink interface in the same way as the PDRingress queuing discipline does. It will also inform the process if a PDR_Release_Request is received. In this implementation, the egress daemon will listen for that event. The queuing discipline has one single class but no filters, exactly like the PDRingress queuing discipline.

4.4.2.6 The PHRegress Queuing Discipline

This queuing discipline is responsible for catching PHR and PDR signaling messages for new flows. It will inform the egress daemon via the rtnetlink interface about all PHR and PDR messages and not forward them to the next hop outside the domain. Even this queuing discipline has only one single class. That class will hold the queuing discipline for best effort traffic.

4.4.2.7 The Pipe Queuing Discipline

This queuing discipline connects two queuing disciplines after each other and implements some kind of virtual wire with infinitive bandwidth between them. It has two classes, where packets are enqueued on the first class and dequeued from the second. As many packets as possible will be forwarded from the first class to the second. This is done all the time by dequeuing packets from the queuing discipline associated with the first class and enqueuing them on the queuing discipline associated with the second class. No filters can be added to the pipe queuing discipline.

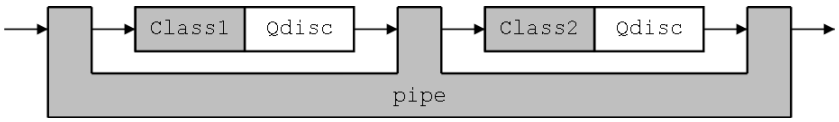

Figure 4-10: The pipe queuing discipline

4.4.2.8 The Ingress Daemon

The ingress daemon is responsible for the message interaction with SEP and mappings between SEP and the RODA PHR. It is implemented in the ingress node and inspects SEP packets with the IP router alert option, in the same way as any SEP-aware router. When a SEP packet is received, it does basic SEP admission control and installs the new queuing disciplines on the correct outgoing interface. When the flow is released, it will remove the queuing disciplines again. The ingress daemon has per-flow states and keeps track of each flow, so that it, for instance, can send a SEP Tear Down Flow packet if a PHR message is marked. The daemon manages the PDRingress queuing disciplines in the kernel and listens for their messages.

4.4.2.9 The Egress Daemon

The egress daemon processes the events from the PHRegress queuing disciplines. When a PDR_Reservation_Request with a PHR_Resource_Request packet is received, it will perform egress admission control. If a flow is accepted, it will create a PDRegress queuing disciplines for the new flow, send back a PDR_Reservation_Report message and finally forward the SEP request to the next hop. It manages all the flows in the egress node and their PDRegress queuing disciplines in the kernel. It listens for timeouts and release messages. If a PDR state timeouts or is released, it will remove the flow's PDRegress queuing disciplines from the kernel.

4.4.3 Ingress Node Implementation

This section describes a node that only serves flows as an ingress node. As was said in section 4.3, the traffic control in Linux is located on the outgoing interface. In an ingress node, this interface needs to do two things.

It must do per-flow scheduling and police each entering flow independently. Each flow must never receive more resources than it reserved, otherwise it may disrupt the quality of the other flows. Secondly, it must perform Diffserv packet scheduling and the PHR functionality per traffic class. Figure 4-11 shows how the traffic control inside the kernel looks like and how the ingress daemon interacts with it. From now on, the classes of classful queuing disciplines will not be explicitly showed in the figures, so that the framework will be smaller and easier to understand.

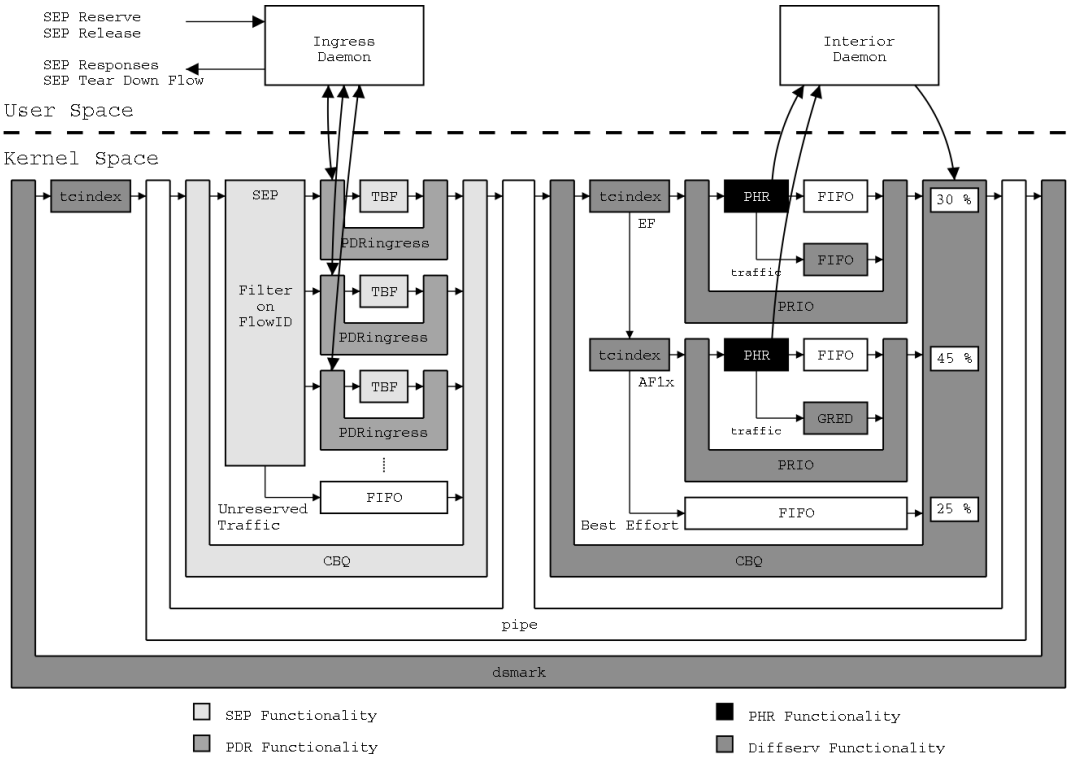

Figure 4-11: The outgoing interface scheduling framework in an ingress node

Packets arriving from outside will first enter the left half of the queuing structure in Figure 4-11. There they will be classified by the SEP filter and enqueued on one of the PDRingress queuing disciplines if they belong to a reserved flow. If not, they will be placed on the unreserved traffic queue. If a packet is placed on the unreserved traffic queue, it will be marked as best effort and just forwarded to the right part of the figure for packet scheduling. If it belongs to a reserved flow, it will be marked by the PDRingress queuing discipline and then, shaped and policed by a token bucket filter, denoted TBF in Figure 4-11.

The right part of Figure 4-11 is a packet scheduling part that works more or less the same as a Diffserv node. It performs packet scheduling per traffic class, as any normal Diffserv node, but with the addition of the PHR filter described earlier in this chapter. The figure shows three traffic classes, one for EF, one for AF and one best effort. Dynamic provisioning using the RODA PHR is used in the EF and the AF traffic classes.

The right part first classifies the packets, based on the DSCP, into their traffic classes. If the traffic class uses the RODA PHR resource management, a filter will separate signaling packets from user data traffic. The signaling messages will first be processed in the PHR filter and then placed in a FIFO queue with higher priority than the data traffic. The PHR filter stores the PHR reservation state for the traffic class it is associated with. The sliding window algorithm, as explained in section 4.4.1, is implemented in the PHR filter and it performs the marking of PHR messages. The PHR filter is configured with the refresh period, the number of cells per window and a threshold, during the initialization process.

The ingress daemon listens for SEP packets that have the IP router alert option, using the socket interface and the socket option described in section 4.3.3. If the daemon admits a new flow, it will create new PDRingress and TBF queuing disciplines as in Figure 4-11. It updates the SEP filter with the flow identifier and connects it to the PDRingress queuing discipline. The PDRingress queuing discipline sends a PDR and PHR packet requesting resources for the flow in the Diffserv domain. When the PDR report for the request arrives, the PDRingress queuing discipline lets the data traffic pass through and the sending host can start to send data. The daemon is responsible for all flows and monitors all PDRingress queuing disciplines and their status. They may inform the daemon about events such as timeouts on requests and marked reports. In that case, the ingress daemon sends a SEP Tear Down Flow packet to the sender. It will send a SEP Reservation Rejected if the marked message was a request for a new flow. When the sender releases the flow, due to a tear down or not, the ingress daemon removes the flow identifier from the SEP filter and removes the TBF and PDRingress queuing disciplines. From then on, packets belonging to the old flow will be placed in the unreserved traffic queue.

In the prototype, the traffic classes were assigned a fixed slice of bandwidth using a configuration file. However, it is possible for traffic classes to borrow bandwidth from each other. One way is by using the borrowing concept in the CBQ scheduling algorithm. Another way is to use a daemon for monitoring the long-term characteristics of the traffic classes. The daemon, called interior daemon in Figure 4-11, monitors all classes for their utilization and if necessary, it will re-configure the bandwidth assignments so that highly utilized traffic classes get resources from lightly used traffic classes. At the same time, the daemon must update the RODA PHR thresholds, so that they conform to the new situation.

The interior daemon was not implemented, so the concept has not been tested yet. Anyway, there are also other ways of achieving dynamic resource assignments between traffic classes. In another solution, there is no longer a threshold associated with each traffic class independently, but all together. The daemon monitors the PHR reservations in the traffic classes and re-configures the bandwidth assignments when necessary.

4.4.4 Interior Node Implementation

A pure interior node only needs to implement the right part of Figure 4-11. It does not need to perform any policing, marking, shaping or other SEP functionalities. It only needs to perform packet scheduling per traffic class, according to Diffserv and PHR functionality. However, that functionality is exactly the same as in the ingress node implementation. The outgoing interface in an interior node is shown in Figure 4-12.

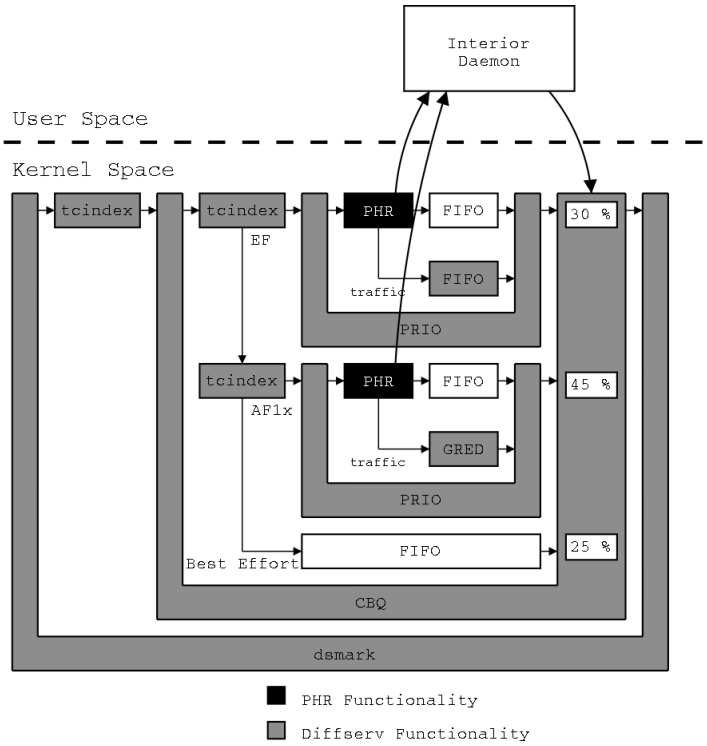

Figure 4-12: The outgoing interface scheduling framework in an interior node

This should explain why the RMD scales well. It is as good as Diffserv, since the PHR functionality is so negligible. In comparison with the Diffserv example shown in Figure 4-8, the only addition is the PHR filter, which implements the sliding window algorithm and the priority class for PHR signaling messages.

4.4.5 Egress Node Implementation

The egress node terminates the PHR and PDR signaling. It also performs per-flow scheduling, since the flows should be shaped before transmitted to the next hop. The SEP filter, once again, classifies the packets and then performs shaping using the TBF queuing discipline. The queuing structure on the outgoing interface of an egress node is shown in Figure 4-13.

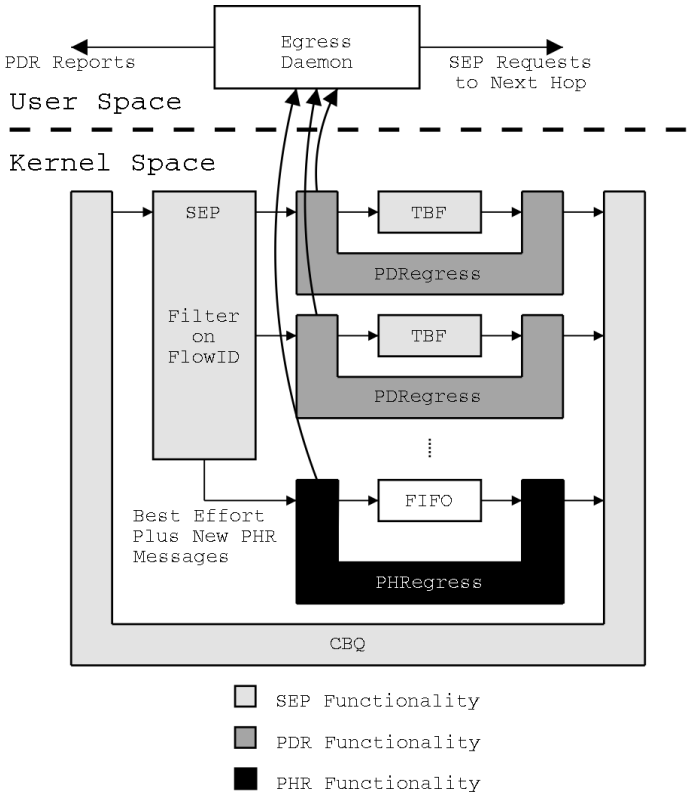

Figure 4-13: The outgoing interface scheduling framework in an egress node

There is one PDRegress queuing discipline per flow and it catches the PHR and PDR messages for that flow and sends back PDR reports to the ingress node. It informs the egress daemon if the flow is released or if it is not refreshed properly. If so, the egress daemon will then remove the PDRegress and TBF queuing disciplines and remove the flow identifier from the SEP filter. If the flow was terminated with a PDR_Release_Request message, the egress daemon will also send a SEP Release message to the next hop.

The best effort traffic and signaling messages belonging to new flows will be classified into the last CBQ class. There, the PHRegress queuing discipline catches all PHR and PDR messages and sends them to the egress daemon. If a request for a new flow is received, the egress daemon will perform egress admission control, and if successful, it creates the PDRegress and TBF queuing disciplines for the new flow. Then, it will add the flow identifier to the SEP filter and connect it to the PDRegress and TBF queuing disciplines. After that, it sends the SEP Request to the next hop and a PDR_Reservation_Report to the ingress node.

The resource management on the outgoing interfaces of egress nodes can be based on the external protocol mechanisms, as in Figure 4-13. In some cases, it may be better to use the RODA PHR resource management or both. Then it is possible to connect the PHR and Diffserv part with the egress part in the same way as on the internal interfaces, but in the opposite order. The PHR and Diffserv part goes to the left and the egress part to the right in the pipe queuing discipline.

The resource management on the outgoing interfaces of the egress nodes can be based on the external protocol mechanisms viewed in Figure 4-13. In some cases, it may be better to combine the external protocol mechanisms with the RODA PHR resource management mechanisms, shown in Figure 4-12. It is then possible to connect the PHR and Diffserv parts

with the external protocol mechanism, by using a pipe queuing discipline similar to the one used in Figure 4-11. However, the PHR and Diffserv parts must be placed at the left part of the pipe queuing discipline, while the external protocol mechanisms should be placed in the right part of the pipe queuing discipline.

4.4.6 The All-round Node Implementation

Implementing three different types of nodes is not very intelligent. In fact, all edge nodes can act as both ingress and egress for different flows at the same time, so they need to support both roles. Moreover, the distinction between edge and interior nodes is only conceptual. Figure 4-14 explains why.

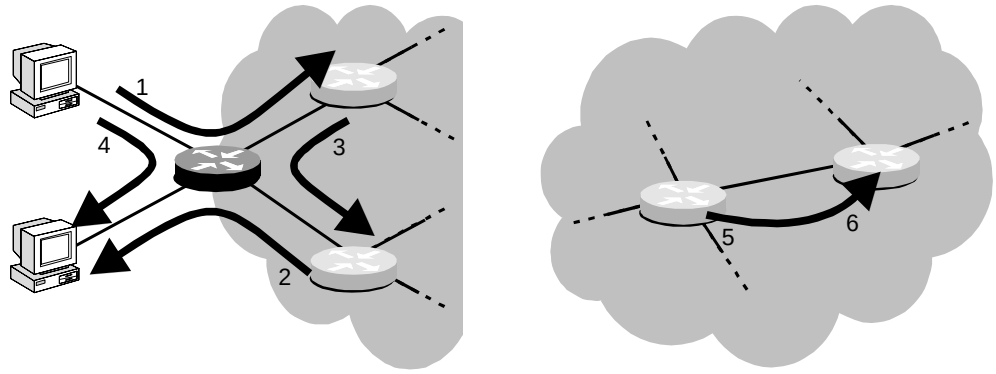

Figure 4-14: Flows through an RMD node

The edge node in the left hand figure has two external links and two internal and can therefore serve four different kinds of flows. First, it can be the ingress node for flows coming from outside and going into the domain. Secondly, it can be the egress for flows going out of the domain. Thirdly, it may also act as an interior node for flows coming from one interior node and going to another interior node. Finally, it may act as an outside node for flows just touching the edge of domain. The last kind of flows do not interact with the RMD and Diffserv functionality at all, they should get the usual SEP behavior.

In the right hand figure, one interior node has a flow to another interior node. There might be applications running in the interior nodes that want to reserve resources for flows. If this is the case, the node must act as an edge node for the flows to and from the applications. For sending applications, the node must act as an ingress node, and for receiving applications, the node must act as an egress node. Therefore, all nodes that belong to the domain must implement and perform all the functionality described in previous sections. However, the prototype can only support the ingress, egress and interior roles, so flow four, five and six in Figure 4-14 cannot be served by the current prototype.

As explained in section 4.3, the traffic control in Linux is implemented on the outgoing interface. The ingress functionality is also implemented on an outgoing interface. Therefore, the outgoing interfaces of a node are divided into two types:

- Internal interfaces. These are the outgoing interfaces going to nodes belonging to the same Diffserv domain. Thus, these interfaces handle traffic being forwarded within the domain and traffic entering the domain.
- External interfaces. These are the outgoing interfaces going to nodes outside the domain. Thus, these interfaces handle traffic leaving the domain and touching flows.

In the example in Figure 4-15, there are two flows passing through a Diffserv domain with RMD and one flow outside the domain. The outgoing interfaces in the nodes are marked with white squares and circles. The middle RMD node acts as interior node for flow one and as ingress node for flow two. The internal interface denoted A in the figure has to support both these two flows.

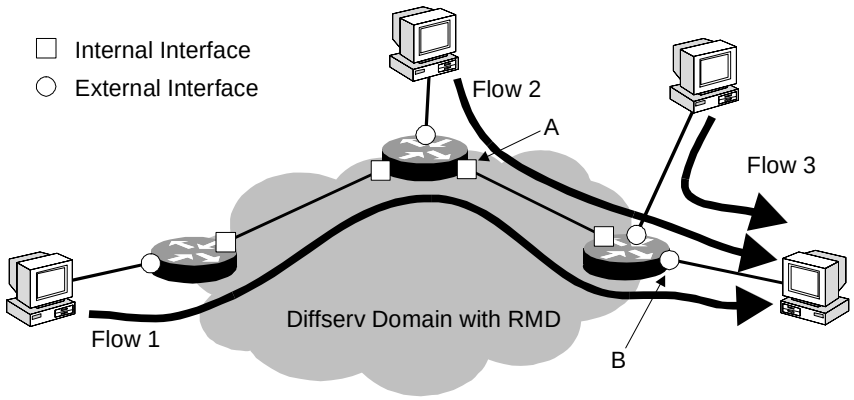

Figure 4-15: RMD nodes and their interfaces

The external interface on the right node, called B, will perform egress functionality for both flow one and two. It will also handle the touching flow number three. Therefore, the outgoing link must use the SEP scheduling and admission control mechanisms. In the touching flow case, policing may also be necessary.

The flows on the external interface are all SEP flows, but only the flows leaving the domain, like flow one and two, are passing the PHR reservation state maintenance algorithm. Therefore, it is necessary to use the SEP admission control functionality on the external interface. Actually, this is the only necessary admission control, since it can be more precise than the RODA PHR resource management. However, under some circumstances, it might be necessary to add the PHR resource management functionality even on the external interfaces.

4.4.7 Traffic Control on the Internal Interfaces

Every internal interface needs the Diffserv and PHR functionality, since the link resources must be managed by the RODA PHR. If the node at the same time has external interfaces, it means that some flows may enter the domain in the node and be routed to an internal interface. Therefore, the internal interface also needs to do per-flow policing and shaping for entering flows. The dual role problem is solved by introducing a sneak entrance to the right part of the pipe queuing discipline into the framework of an ingress node, shown in Figure 4-11. This is implemented by adding a device filter and a short FIFO queue. Figure 4-16 shows how.

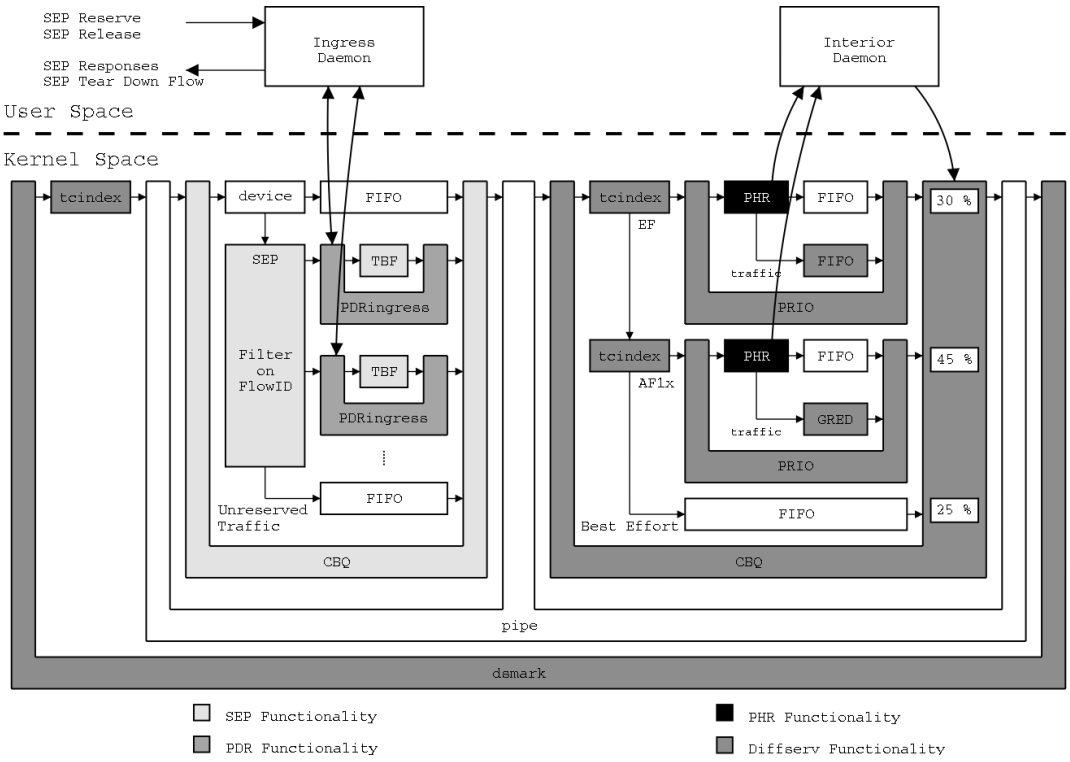

Figure 4-16: Scheduling framework on internal interfaces

The device filter is configured in such a way, that traffic arriving from nodes inside of the domain is matched. Such traffic will be directly forwarded to the interior scheduling part, such as flow one on interface A in Figure 4-15. Traffic coming from external nodes must first be classified, policed, marked and shaped before entering the domain, such as flow two on interface A. In Figure 4-16, these flows do not match the device filter and is filtered by the SEP filter into its class. Best effort traffic coming from outside does not match, neither the device nor the SEP filter, and is therefore placed in the last FIFO queue as before. Otherwise, everything is the same as the ingress only node.

4.4.8 Traffic Control on the External Interfaces

The external interfaces need to support both the egress functionality and the touching flows. This means that the external interfaces have more or less the same egress-scheduling framework as in Figure 4-13. The only addition is the support for touching flows, such as flow three on interface B in Figure 4-15. These are handled in the same manner as flows leaving the domain, but without the PDRegress queuing discipline. Figure 4-17 shows how external interface B, in Figure 4-15, looks like for the three flows.

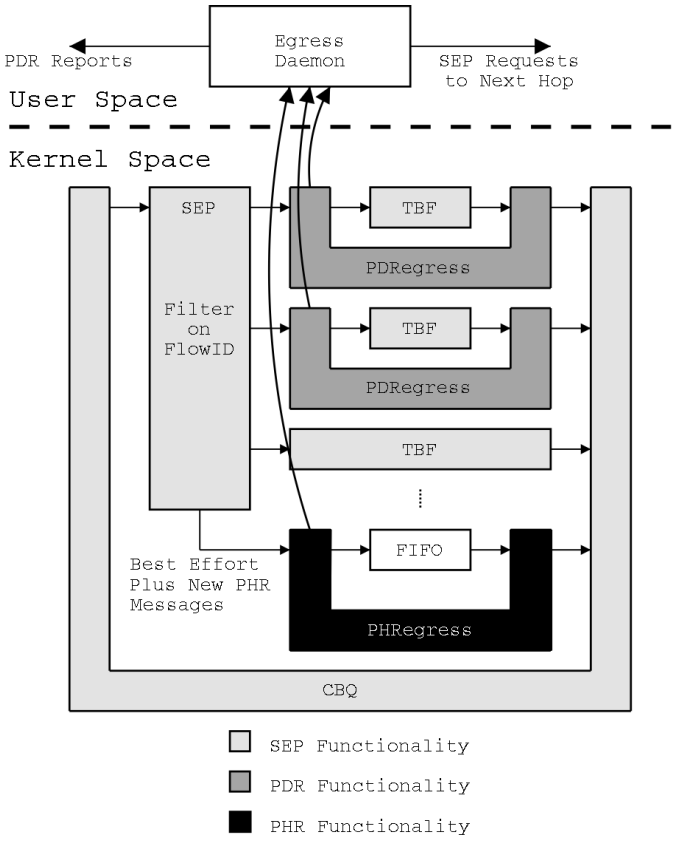

Figure 4-17: Scheduling framework on external interfaces

The touching flows are handled by the ingress daemon in the same way as if the node was an ordinary SEP-aware node. An all-round node is namely running both the ingress and egress daemons. The ingress daemon will handle the SEP Reserve packet and will find the correct external outgoing interface. It will add a TBF queuing discipline for shaping and connect a flow identifier in the SEP filter to this queuing discipline. However, it will also tell the SEP filter to perform policing on the flow. The TBF filter without a PDRegress queuing discipline in Figure 4-17 is a touching flow.

4.5 Testbed Implementation

To be able to test the RMD implementation, it is necessary to have a SEP-aware application that can generate requests. It is also necessary to have a SEP daemon that can reply to the SEP requests in a receiver host. Three small tools were developed to do this. One simple tool located in the receiving host that answers all SEP Reserve packets with a SEP Reservation Accepted reply. A second tool located in the sending host that generates SEP Reserve packets and starts to send data traffic. A third tool also located in the sending host that generates many SEP flows and releases them according to a Poisson arrival process. All three tools were written in Python.

For generating the data traffic, the iperf program [TiFe] was used. It was modified so that it sends user data using a specific source port number. This is necessary since we first need to send the SEP request, with source address, protocol and port number and then start iperf. The iperf sends UDP data during some time and then prints statistics, such as delay, jitter and packet losses.

[5] EXPERIMENTAL RESULTS

This chapter describes the experiments and the demonstration that were carried out with the RMD prototype implementation. The goal of the prototype was to emphasize that the RMD functionality can be implemented in the Linux environment. The experiments performed in this section were specified to show that the RMD implementation can support the expected requirements. If the RMD prototype does not behave as expected, then the results of the experiments will be used as feedback information to enhance the future RMD design.

This chapter introduces the test environment and the configuration setup in each node. Thereafter, the experiments are described with screenshots and discussions. Finally, the demonstration is described.

5.1 Test Configuration

The experiments and the demonstration were all accomplished in the network depicted in Figure 5-1. The sender generated SEP flows to the receiver, using UDP and port numbers starting from 5000. The receiver ran the SEP daemon discussed in section 4.5 and always accepted the SEP Reserve packets by sending a SEP Reservation Accepted packet back. Only the RMD-enabled Diffserv domain was able to reject the flows. All the links were 100 Mbps point-to-point Ethernet connections. The nodes were PCs of various models and all ran the RMD prototype implementation, which was, at the time, based on the Linux 2.4.1 kernel.

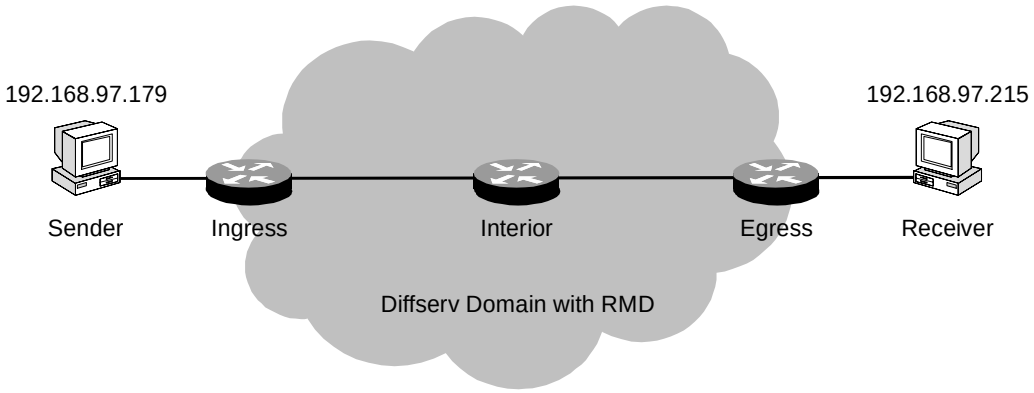

Figure 5-1: Test configuration

The sender application created reservations based on a Poisson arrival process. During the experiments, the average inter-arrival time was three seconds and the average holding time for the flows was 100 seconds. All flows reserved always a rate of 48 kbps or 6000 bytes per second. With that reservation, the application could send 4 packets per seconds with a packet

length of 1500 bytes. The token bucket size was chosen to be 15000 bytes, which is the same as 10 packets of the maximum size.

Parameter	Value
Unit	8 kbps (1000 Bytes per second)
Refresh Period	10 seconds
Window Size	5 cells
Unit Threshold	100 units (800 kbps)
Signal Timeout	1 second

Table 5-1: Configuration values

Table 5-1 shows the configuration that was used by the RMD-enabled Diffserv domain in the experiments and the demonstration. There were two traffic classes defined, one for best effort traffic and one for reserved flows. The latter uses the EF PHB and a queue length of five packets. The best effort class was given 1 Mbps and the EF class was given 800 kbps. The rest of the 100 Mbps were unused and could not be utilized by anyone.

In the EF class, within the Diffserv domain, one unit corresponded to 8 kbps. This means that the reservations were translated into 6 units each and these units must be refreshed by a PHR_Refresh_Update every 10th second.

The signal timeout is the time that the ingress node should wait, after that a request message is sent until the request is considered lost. If a report message associated with the request message that initiated this signal timeout period is received by the ingress node within the signal timeout, then the ingress node deduces that the request message has been successfully received by the egress node. Otherwise, the ingress node deduces that the request message was lost. The round trip time for a PDR or PHR signaling message between two edge nodes has to be smaller than the signal timeout, otherwise signaling messages can accidentally be considered as lost by the ingress, even if they are not.

5.2 Experiments

This section describes and shows four different experiments that were performed using the RMD prototype. The depicted screenshots show the output from the ingress daemon, the egress daemon and a graph of the RODA PHR reservation state for the EF class in the interior node. The two first presented experiments show normal operation and the last two show some fault handling operations.

5.2.1 Successful Reservation

In the first experiment, the sender generated seven flows by sending seven SEP Reserve packets towards the receiver. The sender also received the seven SEP Reservation Accepted packets since all 100 units were available when the experiment began. The output of the two daemons and the graph tool is shown in Figure 5-2.

Figure 5-2: Screenshots of the prototype showing seven successful reservations

The top window shows the RODA PHR reservation state history of the interior node. It clearly shows that 42 units, or seven flows with 6 units each, was reserved at the time of the snapshot. The Y-axis of the graph shows the number of reserved units and the X-axis of the graph represents the time. The left window shows the output of the ingress daemon. It has successfully registered the seven flows from the sender, which used IP address 192.168.97.179. The active flows, beginning with source port number 5000 and ending with number 5006, are listed in the upper sub-window of the ingress daemon window. The lower sub-window of the ingress daemon shows the registered events in chronological order. Note that no calls have been rejected and all of them are listed as active flows.

The right window shows the egress daemon and that window is also divided into two sub-windows. The top sub-window lists the active flows registered by the egress daemon and the lower sub-window shows the events, in the same way as the ingress daemon. The screenshot shows that also the egress daemon has registered the seven flows.

In RMD, the per-flow information is only used in the edge nodes. The interior node is not shown in the figure since no per-flow information is available there. Only the RODA PHR reservation state is used in the interior node, and the history of that value is shown in the top window.

5.2.2 Unsuccessful Reservation

In this experiment, the generation of the flow was allowed to continue for some time. Since the average inter-arrival time was three seconds and the average holding time for the flows was 100 seconds, all resources became reserved quite fast. This means that the interior node

started to mark some of the PHR_Resource_Request messages. Figure 5-3, similar to Figure 5-2, depicts a snapshot of the program at that point. The top window shows the RODA PHR reservation state history in the interior node. The left bottom window shows the output of the ingress daemon and the right bottom window shows the output of the egress daemon.

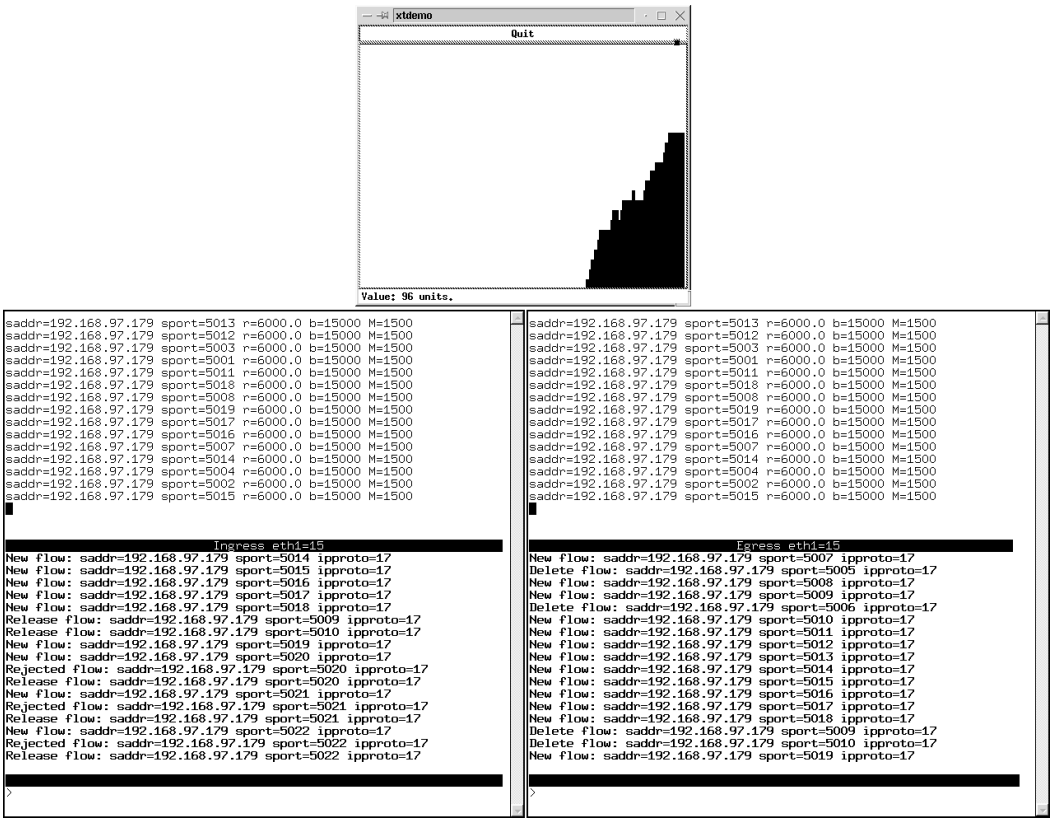

Figure 5-3: Screenshots of the prototype showing normal operation

The event window of the ingress daemon shows that it first admitted the flows with source ports 5014 up to 5017 by printing the following lines:

```
New flow: saddr=192.168.97.179 sport=5014 ipproto=17
New flow: saddr=192.168.97.179 sport=5015 ipproto=17
New flow: saddr=192.168.97.179 sport=5016 ipproto=17
New flow: saddr=192.168.97.179 sport=5017 ipproto=17
New flow: saddr=192.168.97.179 sport=5018 ipproto=17
```

After that, two flows were released by the sender. This led to the following two lines in the egress daemon:

```
Release flow: saddr=192.168.97.179 sport=5009 ipproto=17
Release flow: saddr=192.168.97.179 sport=5010 ipproto=17
```

Then the sender tried to reserve two more flows, but only the first one was accepted. The second one was rejected because an interior node marked the packet. The ingress daemon prints that it rejected a flow and then sends a SEP Reservation Rejected packet to the sender. In response to the rejection, the sender sent a SEP Release packet and that event is also logged by the ingress daemon. This is what the ingress daemon printed because of this:

```
New flow: saddr=192.168.97.179 sport=5019 ipproto=17
New flow: saddr=192.168.97.179 sport=5020 ipproto=17
Rejected flow: saddr=192.168.97.179 sport=5020 ipproto=17
Release flow: saddr=192.168.97.179 sport=5020 ipproto=17
```

At this point, no more flows were accepted. The ingress daemon rejects two more flows before the snapshot was taken. Note that the egress daemon does not register rejected flows at all. It only logs when it installs a new flow and when an active flow is deleted.

The sender tool also logs its actions and events. The following is what the sender logged during these last period of this experiment. The number to the left is the source port number of the flow associated with the log message.

```
[5014] sent Reserve.
[5014] recv Reservation Accepted.
[5015] sent Reserve.
[5015] recv Reservation Accepted.
[5016] sent Reserve.
[5016] recv Reservation Accepted.
[5017] sent Reserve.
[5017] recv Reservation Accepted.
[5018] sent Reserve.
[5018] recv Reservation Accepted.
[5009] sent Release.
[5010] sent Release.
[5019] sent Reserve.
[5019] recv Reservation Accepted.
[5020] sent Reserve.
[5020] recv Reservation Rejected.
[5020] sent Release.
[5021] sent Reserve.
[5021] recv Reservation Rejected.
[5021] sent Release.
[5022] sent Reserve.
[5022] recv Reservation Rejected.
[5022] sent Release.
```

If the process was allowed to continue even further, it is shown that no existing flows will be stopped by a SEP Tear Down Flow packet. Only the new reservations would be rejected. After some time, the resources from the two released flows would have become free and two new flows would have been accepted again.

5.2.3 Lost PHR signaling messages

The third experiment emphasizes the functionality of the RMD scheme when some RMD signaling packets are lost. In this experiment, three flows were generated. The physical cable used to support the communication between the ingress node and the interior node was unplugged to emulate the loss of PHR and PDR signaling messages. Figure 5-4 shows the outputs of the graph tool and the ingress and egress daemons. In particular, it shows how the ingress daemon notices a signal timeout for two PDR_Refresh_Request messages and how the egress daemon notices a soft-state release of the same reservations, since no refresh messages for these flows were received. At the snapshot, the two flows were already released. The third flow survived since the cable was only disconnected for a short time and during that time, it was not necessary to refresh the reservations for that flow.

Figure 5-4: Screenshots of the prototype showing lost signaling messages

The ingress daemon printed "Timeout flow" when the PDR_Refresh_Report was not received within the signal timeout period. At the same time, the daemon sent a SEP Tear Down Flow to the sender, which responded with a SEP Release packet. The flow was released in the ingress node by that packet.

In the egress node, the refresh timer for the two flows expired, so the daemon released the flows. The classid printed in the event window is an internal representation. Classid one refers to the flow with source port 5000 and classid two to the flow with source port 5001.

Only the third flow was active when the snapshot was taken a few seconds after the packets were lost. The graph shows clearly how the reservations were released when no refreshes reached the interior node.

5.2.4 Severe Congestion

The fourth experiment emphasizes the functionality of the RMD scheme when a severe congestion situation occurs in the interior node. Figure 5-5 shows the outputs of the graph tool and the ingress and egress nodes. In this experiment, the interior node was instructed to mark every PHR message with the severe congestion flag. This was done by instructing the PHR filter to do so, with the tc command. The severe congestion situation was then marked in the subsequent received PHR messages. The interior node set the S-flag of all these messages and forwarded them towards the egress node. The egress node responded with a PDR report to the ingress. The ingress node saw the S-flag in the report and activated a severe congestion state, which means that it rejected all incoming calls for some time and at the same time released the flows whose PDR reports were S-marked.

Figure 5-5: Screenshots of the prototype showing severe congestion handling

The ingress node reported the received S-marked PDR_Refresh_Report messages by printing the following line:

```
Flow severe marked: saddr=192.168.97.179 sport=5023 ipproto=17
```

After that, the flow is teared down in the same way as before. However, the egress daemon does not notice anything in the current prototype. This can of course be easily changed in future versions, if it is considered useful.

5.3 The Demonstration

During the demonstration, all experiments, described in the previous sections, were repeated. In addition, also a new demo experiment was accomplished to show how the data traffic is handled for reserved and unreserved flows. A real-time application, called Robust Audio Tool (RAT) [BeCa98], was used for this purpose. It sent speech and music from the sender to the receiver in the network depicted in Figure 5-1. RAT was configured to send audio data at a rate of 100 UDP packets per second, where each packet had a size of 360 bytes, including the IP header. This means a rate of 36,000 bytes per second (4.5 kbps) or 36 units inside the domain. Furthermore, RAT was also configured not to compensate for bad connections and to have a very short receive buffer. The demo experiment was divided into two scenarios.

In the first scenario, no reservation was made for the real-time application and its traffic was therefore classed as best effort class. The audio quality was perfect when the network was unloaded. The sender then also started to generate 2 Mbps background traffic. In the Diffserv domain, these 2 Mbps had to compete with the real-time traffic in the best effort class, but the best effort class had only 1 Mbps of assigned bandwidth. The audience's hearing perception was used to verify the quality of the real-time audio. The audience had no problem in hearing a degradation of the audio quality.

For the second scenario, the background traffic was removed again. Then, a reservation was made for the real-time application. No quality change in the audio was possible to hear when the reservation was in place. The data traffic was marked as reserved and did not compete with the best effort traffic. Adding the background traffic again did not affect the quality of the audio in any way.

[6] CONCLUSIONS

This thesis has described the resource management problem in an IP-based UTRAN. It investigated current resource management schemes for IP networks and concluded that none of them satisfies all requirements by an IP-based UTRAN. Furthermore, it introduced a resource management scheme for Diffserv domains, called Resource Management in Diffserv (RMD). Then, it described a prototype implementation of RMD and listed results from experiments testing the basic functionality of the prototype.

This chapter gives some conclusions from the design of RMD and the implementation of the RMD prototype. It also gives some recommendations and suggestions for future work.

6.1 Conclusions and Recommendations

The RMD framework and the RODA PHR offer a very scalable and precise resource management solution, even in large Diffserv domains. With an RMD-enabled Diffserv domain, it is possible to offer good QoS services, almost in line with the ones offered by Intserv. The RODA PHR together with the sliding window algorithm can achieve a very good approximation of the resource utilization in the network. This makes it possible for Diffserv domains to offer a quantitative resource reservation service. The prototype verifies this by demonstrating that the RMD solution works as expected.

In addition to the RODA PHR, a simple resource reservation protocol was developed that can be used on an end-to-end basis. This protocol, called SEP, was designed to be as simple as possible to implement. Moreover, a PDR protocol, which can interoperate with the SEP protocol and the RODA PHR, was also defined. The PDR, using the RODA PHR and SEP as the external protocol, were implemented in Linux using kernel version 2.4.

Due to its simplicity, the implementation of the interior functionality in the Linux environment was easy to do. The additional parts that were required by the implementation of the interior functionality were only the sliding window algorithm and the PHR packet filter. It is expected that if it will be possible to implement the Diffserv functionality in hardware, it will also be possible to implement the PHR interior functionality in hardware.

However, the implementation of the edge functionality was more complicated. The design does not support any nodes in the domain to be either a sender or receiver for SEP flows. Furthermore, the design of the prototype is specific to the Simple PDR, which was developed from the special characteristics of SEP. To support other PDR protocols, a new design is needed.

Implementing a PDR with support for soft-state external protocols, such as RSVP, might be much more complicated and implementing support for more than one PDR, is probably even more complicated. Therefore, it is expected that it will be difficult to implement the PDR edge functionality in hardware.

6.2 Future Work

Now that the RMD is defined and a working prototype of RMD is constructed, there are two ways to proceed. First, it is possible to try to improve the RMD framework and adjust specifications and parameters. The second way is to try to use RMD in real environments, such as the IP-based UTRAN. Which parameters would be best to use? How to define the unit? How should the traffic classes be chosen and which PHBs should they have? How will RMD and the RODA PHR work in bigger and more complicated network topologies?

The Simple PDR, used in the prototype, is only supporting the limited external protocol, SEP. Therefore, a new PDR protocol for RSVP and Integrated Services should be defined and tested. This PDR should investigate how to support a receiver-initiated external protocol in the best way. Can it be solved without extra signaling loops and how to support external protocols that use soft-state reservations? RSVP is the best candidate to consider, since it is widely spread, receiver-initiated and uses soft-state reservations.

Another future work activity is to define and implement a measurement-based PHR protocol. Measurement-based protocols are interesting, because they do not introduce any states in any node and can therefore be even less complex than the RODA PHR.

A problem with the RODA PHR is that utilization decreases when the refresh period increases. If a flow is released one second after the refresh is sent and the period length is ten seconds, then it will take at least nine seconds before the reservation is released by the soft-state principle. To overcome this problem, a RODA PHR release message can be introduced. However, this is not trivial, since the interior nodes do not have any per-flow states.

Finally, it is also important to understand how the correlation between reserved units and the traffic load works. This may, in some cases, affect the QoS and, in other cases, it may give an under-utilized network. Therefore, it must be investigated how the units should be defined for different PHBs and how the thresholds in each traffic class should be assigned. How does these parameters affect the network utilization and the QoS?

The successful prototype implementation shows that the RMD approach is promising. Its applicability in the IP-based UTRAN is only one example. Maybe one day, RMD will be able to solve most of the resource management problems in the IP networks and be a key link to Internet QoS.

[7] REFERENCES

[3GPP-21905] Zarri, M., *Vocabulary for 3GPP Specifications*, 3rd Generation Partnership Project, TS 21.905, October 1999.

[3GPP-25401] Calmel, J-M., *UTRAN Overall Description*, 3rd Generation Partnership Project, TS 25.401, January 2001.

[AhBe99] Ahlard, D., Bergkvist, J., Cselenyi, I., *Boomerang Protocol Specificatiom*, Internet-Draft, draft-bergkvist-boomerang-spec-00.txt, http://boomerang.ttt.bme.hu/, June 1999.

[AlFe98] Almesberger, W., Ferrari, T., Le Boudec, J.-Y., *SRP: a Scalable Resource Reservation Protocol for the Internet*, vol. 21, no. 14, pp. 1200-1211, September 1998.

[AlHa99] Almesberger, W., Hadi Salim, J., Kuznetsov, A., Differentiated Services on Linux, ftp://icaftp.epfl.ch/pub/linux/diffserv/misc/dsid-01.ps.gz, June 1999.

[Alme99] Almesberger, W., *Linux Network Traffic Control - Implementation Overview*, ftp://icaftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps.gz, April 1999.

[BaIt01] Baker, F., Iturralde, C., Le Faucheur, F., Davie, B., *Aggregation of RSVP for IPv4 and IPv6 Reservations*, Internet-Draft, draft-ietf-issll-rsvp-aggr-04.txt, Work in Progress, April 2001.

[BeCa98] Bennett, R., Clark, L., Hasler, K., User Guide for RAT v4.2, http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/, 1998.

[BeCs00] Bergkvist, J., Cselenyi, I., Ahlard, D., Boomerang – A simple Resource Reservation Framework for IP, Internet-Draft, draft-bergkvist-boomerang-framework-01.txt, http://boomerang.ttt.bme.hu/, November 2000.

[BiBl01] Bianchi, G., Blefari-Melazzi, N., *Per Flow Admission Control over AF PHB Classes*, Internet-Draft, draft-bianchi-blefari-admcontr-over-af-phb-00.txt, Work in Progress, March 2001.

[ChLe99] Chow, H., Leon-Garcia, A., *A Feedback Control Extension to Differentiated Services*, Internet-Draft, draft-chow-diffserv-fbctrl-00.pdf, http://www.comm.utoronto.ca/~keith/pub.html, Work in Progress, March 1999.

[DhSu99] Dhandapani, G., Sundaresan, A., *Netlink Sockets - Overview*, http://qos.ittc.ukans.edu/netlink/html/index.html, September 1999.

[Erik98] Eriksson, A., Resource Reservation in a Connectionless Network, Proceedings of Performance Information and Communications Systems (PICS'98), May 1998.

[FeNe99] Feher, G., Nemeth, K., Maliosz, M., Ahlard, D., Bergkvist, J., Cselenyi, I., Engborg, T., *Boomerang – A simple Protocol for Resource Reservation in IP Networks*, IEEE Workshop on QoS Support for Real-Time Internet Applications, Vancouver, Canada, June 1999.

[FlJa93] Floyd, S., Jacobson, V., *Random Early Detection Gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, vol. 1, no. 4, August 1995.

[FlJa95] Floyd, S., Jacobson, V., *Link-Sharing and Resource Management Models for Packet Networks*, IEEE/ACM Transactions on Networking, vol. 3, no. 4, August 1995.

[GiKe97] Gibbens, R. J., Kelly, F. P., *Measurement-based Connection Admission Control*, ITC 15, June 1997.

[GiKe99] Gibbens, R. J., Kelly, F. P., *Distributed Connection Acceptance Control for a Connectionless Network*, Proceedings of 16th International Teletraffic Congress in Edinburgh, June 1999.

[Kilkki99] Kilkki, K., *Differentiated Services for the Internet*, Macmillan Technical Publishing, 1999.

[Lutz96] Lutz, M., *Programming Python*, O'Reilly & Associates, October 1996.

[MoPa92] Mouly, M., Pautet, M.-B., *The GSM System for Mobile Communications*, Cell & Sys., 1992.

[PaKa01] Partain, D., Karagiannis, G., Wallentin, P., Westberg, L., *Resource Reservation Issues in Cellular Access Networks*, Internet-Draft, draft-partain-wireless-issues-00.txt, Work in Progress, April 2001.

[PaSc98] Pan, P., Schulzrinne, H., *YESSIR: A Simple Reservation Mechanism for the Internet*, Proceedings of the 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 98), July 1998.

[Radh99] Radhakrishnan, S., *Linux - Advanced Networking Overview Version 1*, http://qos.ittc.ukans.edu/howto/index.html, August 1999.

[ReOn98] Reichmeyer, F., Ong, L., Terzis, A., Zhang, L., Yavatkar, R., *A Two-tier Resource Management Model for Differentiated Services Networks*, http://irl.cs.ucla.edu/publications.f.html, November 1998.

[RFC768] Postel, J., *User Datagram Protocol*, IETF RFC 768, August 1980.

[RFC791] Postel, J., *Internet Protocol*, IETF RFC 791, September 1981.

[RFC1157] Case, J., Fedor, M., Schoffstall, M., Davin, J., *A Simple Network Management Protocol (SNMP)*, IETF RFC 1157, May 1990.

[RFC1631] Egevang, K., Francis, P., *The IP Network Address Translator (NAT)*, IETF RFC 1631, May 1994.

[RFC1633] Braden, R., Clark, D., Shenker, S., *Integrated Services in the Internet Architecture: an Overview*, IETF RFC 1633, June 1994.

[RFC1889] Schulzrinne, H., Casner, S., Frederick, R., , V., *RTP: A Transport Protocol for Real-Time Applications*, IETF RFC 1889, January 1996.

[RFC2113] Katz, D., *IP Router Alert Option*, IETF RFC 2113, February 1997.

[RFC2205] Braden, R., Zhang, L., Berson, S., Herzog, S., Jamin, S., *Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification*, IETF RFC 2205, September 1997.

[RFC2210] Wroclawski, J., *The Use of RSVP with IETF Integrated Services*, IETF RFC 2210, September 1997.

[RFC2460] Deering, S., Hinden, R., *Internet Protocol, Version 6 (IPv6) Specification*, IETF RFC 2460, December 1998.

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Wiess, W., *An Architecture for Differentiated Services*, IETF RFC 2475, December 1998.

[RFC2570] Case, J., Mundy, R., Partain, D., Stewart, B., *Introduction to Version 3 of the Internet-standard Network Management Framework*, IETF RFC 2570, April 1999.

[RFC2597] Heinanen, J., Baker, F., Weiss, W., Wroclawski, J., *Assured Forwarding PHB Group*, IETF RFC 2597, June 1999.

[RFC2598] Jacobson, V., Nichols, K., Poduri, K., *An Expedited Forwarding PHB*, IETF RFC 2598, June 1999.

[RFC2748] Durham, D., Boyle, J., Cohen, R., Herzog, S., Rajan, R., Sastry, A., *The COPS (Common Open Policy Service) Protocol*, IETF RFC 2748, January 2000.

[RFC2836] Brim, S., Carpenter, B., Le Faucheur, F., *Per Hop Behavior Identification Codes*, IETF RFC 2836, May 2000.

[RFC2998] Bernet, Y., Ford, P., Yavatkar, R., Baker, F., Zhang, L., Speer, M., Braden, R., Davie, B., Wroclawski, J., Felstaine, E., *A Framework for Integrated Services Operation over Diffserv Networks*, IETF RFC 2998, November 2000.

[Stev98] Stevens, W., R., *UNIX Network Programming, Volume 1*, 2nd Edition, Prentice Hall, 1998.

[StZh99] Stoica, I., Zhang, H., Shenker, S., Yavatkar, R., Stephens, D., Malis, A., Bernet, Y., Wang, Z., Baker, F., Wroclawski, J., Song, C., Wilder, R., *Per Hop Behaviors Based on Dynamic Packet States*, Internet-Draft, draft-stoica-diffserv-dps-00.txt, Work in Progress, February 1999.

[TiFe01] Tirumala, A., Ferguson, J., Iperf Version 1.2, http://dast.nlanr.net/Projects/Iperf/, May 2001.

[WeJa01a] Westberg, L., Jacobsson, M., Karagiannis, G., Oosthoek, S., Partain, D., Rexhepi, V., Szabo, R., Wallentin, P., *Resource Management in Diffserv (RMD) Framework*, Internet-Draft, draft-westberg-rmd-framework-00.txt, Work in Progress, April 2001.

[WeJa01b] Westberg, L., Jacobsson, M., Karagiannis, G., Oosthoek, S., Partain, D., Rexhepi, V., Szabo, R., Wallentin, P., *Resource Management in Diffserv On DemAnd (RODA) PHR*, Internet-Draft, draft-westberg-rmd-od-phr-00.txt, Work in Progress, April 2001

[WeTu00] Westberg, L., Turanyi, Z., Partain, D., *Load Control of Real-Time Traffic*, Internet-Draft, draft-westberg-loadcntr-03.txt, Work in progress, April 2000.

[WhCr98] White, P., Crowcroft, J., *A Dynamic Sender-Initiated Reservation Protocol for the Internet*, Proceedings of High Performance Networking (HPN'98), September 1998.