Appendix B
Network Emulation Focusing on QoS-Oriented Satellite Communication

Laurent Dairaine, Mathieu Gineste, and Hervé Thalmensy

Abstract The rapidly growing Internet architecture is causing most recent computer applications to integrate a more or less important part of distributed functionalities—such as transport layer services, transport protocols and other services—that need to meet user's necessities in terms of functionalities and Quality of Service (QoS) requirements. Emulation platforms are a classical way for protocol and applicative experiments to check if user and QoS requirements are met. They complement the simulation and real network experiments, since they enable to use real implementation of protocols or applications without having a real network deployed for the experiments. This chapter presents the emulation approach in the context of networking experimentation: First, the different possible utilisations of dynamic emulation in the context of networking and protocol engineering are presented. Then, requirements for a general network emulation framework are proposed. Furthermore, different network emulation platforms and tools implementing the general framework are exposed; we describe how to use them in the context of protocol engineering and discuss their advantages and disadvantages. Finally, the emulation of wireless systems is challenging, due to many parameters affecting the resulting behaviour of the channel. Satellite emulation, a subset of wireless emulation, has unique characteristics concerning the access to the resource that combines static and dynamic assignment. As an example, the emulation of a QoS-oriented satellite system is detailed in a final section.

B.1 Network Emulation Basics

B.1.1 Introduction to Network Emulation

Rapid growth in Internet network support is leading to distributed computer applications, which not only make use of transport layer services by well-known transport protocols such as TCP or UDP, but also have to implement specific application and/or transport protocols in a way to meet user's necessities in terms of functionalities and QoS requirements. But these applications, and particularly their distributed components, have to be carefully tested to determine whether they perform well, are reliable and robust. This involves the expensive task of testing and debugging the produced software that is expected to run on distributed computers, interconnected by either a large area network such as the Internet, or by specific network technologies like wireless or satellite networks.

Nowadays, end users requirements and competition between software development companies have continuously shortened the development process. But even if the time-to-market decreases, a company cannot release a product if it has not been fully tested and certified to meet clients' expectations in terms of services, QoS and reliability. If the client is disappointed about the product, he will switch to products offered by other companies. These are the reasons why companies need a quality lab that allows the rapid testing of their products under various conditions. There are two classical ways to achieve these tests: simulation and live testing.

Event-driven simulation is a powerful tool to achieve economical and fast experimentation. Its main specifications rely on an ad-hoc model working with and using a logical event-driven technique to achieve the simulation. The modelling techniques allow simplifying the studied problem in order to concentrate on the most critical issues. In addition, the model execution is linked to logical time, not real-time. Depending on the complexity of the model, logical time can be related to real-time. For example, it is possible to simulate a logical hour in few real-time milliseconds or a logical second in several real-time days. Due to both characteristics, no real implementations involving man-in-the-loop are directly possible to be achieved in such simulations. Thus, a reliable link between the tested protocol model and the real implementation has to be accomplished in such a way as to ensure that the offered services and performance results between the simulation model and the experimental implementation protocol are consistent.

Another classical way of testing and debugging distributed software is to use real technology. This technique is generally expensive and time-consuming. The software can be tested on an ad-hoc test-bed using real equipment and implementing a specific technology. This approach is particularly expensive in the context of large area networks, but also in specific technological conditions such as satellite networks. Moreover, it is sometimes impossible to implement this testing method, because the technology support is not available (e.g., developing an application over a new satellite transmission technology). This high experimentation cost is not only due to the technology costs involved, but also to the distributed man-in-the-loop manipulations and synchronisation it requires. Furthermore, it has limited value due to the inherent discrepancies between a particular test network and the much broader range of network imperfection that will be encountered by the software users. Another possibility to experiment with real technology is to use, when possible, the target operational network, e.g. the real Internet. However, lack of control on the network experimentation conditions makes it very difficult to achieve and reduce measurements and tests results' relevance.

More recently, progress in high-speed processing and networking has fostered the rapid development of network emulators. Network emulation is used to conduct experiments implementing real protocols, distributed applications and network models. This enables the network emulator to create a controlled communication environment, which can produce specific target communication behaviour in terms of QoS. Therefore, network emulation reproduces not only real underlying network architecture, but also artificial network impairments aiming at testing the characteristics of the experimented protocol. These tools represent a network service and reproduce the network performance dynamics. This approach is considered to be efficient and useful for simulations and live experiments, because it mixes real-time aspects of the experiment, real and simulated functionalities and emulation model. It provides a means of experimentation using real data and a network model.

B.1.2 What is Network Emulation?

In the most general sense, an emulator is designed to mimic the functions of a system on another system that is potentially totally different - the two different systems should then behave similarly. In software engineering, as shown in Fig. B.1, the same real software can be executed in the same way and without any modification, either in the real system environment or in an emulated environment with a system emulator acting exactly like the real system.

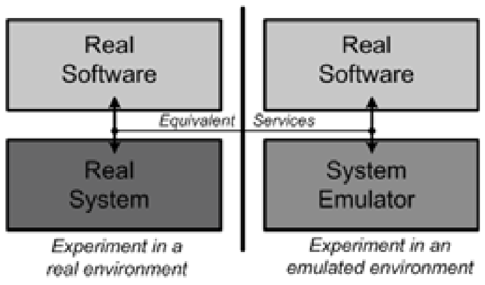

Fig. B.1 Experiment in real or emulated environments.

B.1.2.1 System Emulation and Virtual Machines

A perfect system emulator would offer the same services (functional properties) and the same performance level (non-functional properties) as the reproduced system. In electrical engineering, the word ”emulation” traditionally means a very low-level reproduction of real life electrical signals. For example, professional microprocessor emulator software comes with a processor-shaped connection, which you can actually plug in to a motherboard and run instructions with it. But emulation can be declined into a larger set of applications, for instance various levels of computer systems. Network emulation is an example of a system emulator, where the system is a computer network offering various communication services and is based on a set of well-known protocols.

An example of a system emulator is the VMWare product [185]. The VMware Workstation is desktop virtualisation software for software developers/testers who want to streamline software development, testing and deployment. The VMware Workstation enables users to run multiple x86-based operating systems, including Windows, Linux, FreeBSD, and their applications simultaneously on a single PC in fully networked, portable virtual machines.

The VMware Workstation works by creating fully isolated, secure virtual machines that encapsulate an operating system and its applications. The VMware virtualisation layer maps the physical hardware resources to the virtual machine's resources, so each virtual machine has its own CPU, memory, disks, and I/O devices, and is a full equivalent of a standard x86 machine. VMware can be installed on the host operating system and provides broad hardware support by inheriting device support from the host.

The main purpose of the VMware Workstation is streamline software development and testing, which enhances productivity by e.g. configuring and testing desktops and servers as virtual machines before deploying them to production, and facilitating computer-based training and software demos by deploying classroom material in virtual machines. The VMware workstation also provides virtual networks, allowing the connection between the virtual workstation and the real world. Thus, it is possible for the virtual workstation to communicate with the real host, in which the virtual machine is executed with external real machines, using the physical interface of the host executed. Finally, several virtual machines can also communicate between each other using a virtual network inside the real machine. It is also possible to emulate various virtual networks by interconnecting different virtual machines. Therefore, it is more likely to build a realistic network architecture that implements various subnets with switches, interconnected by virtual hosts implementing the routing functionality.

The VMWare networking capability provides a realistic and functional interface, which allows real pieces of software such as operating systems including networking code (IP, TCP, etc.) and any applications to be executed. It is then possible to build a complex distributed architecture with numerous real implementations of protocols and test the functional characteristics of this system in a best effort environment. Furthermore, performance (e.g., throughput, delay, etc.) will mainly depend on the capacity of the real machine hosting the emulation.

No real-time control is associated with the emulation. The non-functional characteristics of the emulated systems are not supported by the software. The system does not provide a way to represent the QoS characteristics of the target network (and the system), even simplest ones such as a given probability of packet loss, or a simple delay between the end-systems. Of course, more complex network emulation scenarios such as the behaviour of a satellite link with particular atmospherical conditions would not be supported.

B.1.2.2 Network Emulation

While the system emulation allows the users to have a system behaviour on hardware architecture (like Linux on Windows, or hand-held video game systems on computer), network emulation enables network behaviour to be reproduced in a controlled test bed. The main aim of using such emulation tools is to achieve experiments with real applications as well as protocols and provide them with a support ability to evaluate their functional (does the protocol work?) and non-functional properties (how does the protocol performs in particular network conditions?).

The network behaviour can reproduce real technologies (e.g., wireless, satellite link, or network interconnections) as illustrated in Fig. B.2, where the satellite link behaviour is reproduced by a simple emulation box.

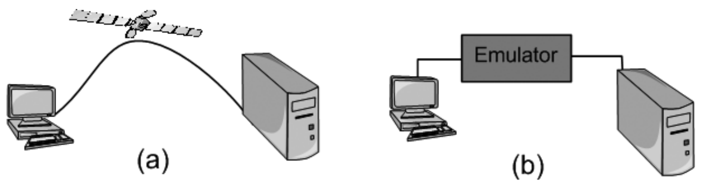

Fig. B.2 Experiment in (a) operational real environment and (b) emulated environments.

Network emulation is used as a method to test non-functional properties of really implemented protocols. It means that emulation tools must provide a way to introduce network impairments such as delay, packet losses, and bit errors according to a model to test the protocols and applications properly. Another important issue about network emulation is to provide a way to produce a predefined possible behaviour to test and stress the experimental protocol in a specific condition.

In the particular but very common case of level-3 IP service emulation, the resulted QoS is mainly dependent on external factors, such as underlying technologies, interconnection topology, network traffic, etc. Thus, IP network service emulation could offer a ”perfect” QoS channel to a 100% loss channel depending on the underlying protocols and many other external factors. This leads to a large set of possibilities in the emulation experiments. The level-3 service emulation is able to produce an end-to-end QoS channel that could focus on:

- Artificial QoS:

The emulator implements processing that aims to test the experimental protocol on specific QoS conditions, not imperatively related to any technology. This processing allows the user to test and stress its experimental protocol in a target QoS condition, aiming to point out errors or bugs that could be difficult to produce in a non-controlled environment. This can be useful for example at transport level to study the impact of various packet drops in a TCP connection (e.g., the SYN/ACK, etc.), or at application level to study what happens if a particular block of the ”Intra” picture is delayed.

- Realistic QoS:

The emulator implements a process that aims to reproduce the behaviour of a specific network architecture as accurately as possible. This type of emulation allows the user to test their protocols over an existing network or internetwork without using a real test bed with all related technologies (e.g., a wireless network, a satellite network, Ethernet Gigabit network, or any interconnection of such technologies).

B.1.3 Why Using Network Emulation?

What types of users are potential customers of an emulation system? Various types of customers with their own aims provide different use cases for emulation systems. Emulation platforms can meet the expectations and help at various stages of protocol research and development:

- At research stage to study the existing solutions and to help the design of new protocols and applications;
- At conception and development stages to study advantages and drawbacks of new proposed solutions compared to the existing ones;
- At test and study of performance stage to test, benchmark, and evaluate the protocols and application;
- At the final stages, to demonstrate the effectiveness of new solution by a real demonstration.

B.1.3.1 Network Emulation at Research and Design Stage

The application and protocol that researchers are working on is for existing or future network solutions. Therefore, a test bed that can help study different cases is required. The emulation can then be used for:

- Precise study of a protocol or an application under specific network conditions to find outside effects, limitations, bugs, or any problems. The advantages of this protocol can also be investigated. These studies will help in proposing new solutions, possibly based on existing ones.
- Comparison of several protocols under specific network conditions to see the advantages and drawbacks of each.
- Comparison of several protocols under realistic network conditions (e.g., satellite network QoS) to reveal the limitations of a solution over some QoS.

B.1.3.2 Network Emulation at Conception and Development Stage

The designer and implementer needs to study his current development to see whether it is reliable and efficient. Different tests can be designed, for example:

- Testing the behaviour of the solution under specific network conditions to find new solutions that can be used to detect potential bugs. Moreover, the test can also be used to investigate the advantages of the new solution under conditions where another solution is deficient.
- Testing the behaviour of the developed solution when a parameter (e.g., delay) is involved. This can help to design charts with the ideal conditions for the solution.
- Testing the behaviour of new solutions under specific network conditions to see whether it can be used on every network or if it cannot be released over a specific technology.
- Doing comparison charts between several protocols and applications, including the new one, to show the pros/cons of the new one.

Live testing would be definitely too complicated and expensive to meet all these conditions. The test phase would therefore be too long, or the tests could be incomplete.

B.1.3.3 Network Emulation at Testing and Performance Evaluation Stage

The end of the testing and development process will be part of the support phases to lead the following experiments:

- Product testing over a realistic end-user network condition to see whether the product can be released to the public. This test is very important to ensure the service provided is adequate with regard to user or application requirements.
- Debugging a product to find why it is not working on a specific network or under specific conditions. It can also provide the proof that the product is well designed for the network, but that presents several bugs under specific end-user computer configurations.

B.1.3.4 Network Emulation when Demonstrating Software

Demonstration is an important activity to adopt the application or the protocol. The protocol and application designer must often demonstrate the efficiency of the proposed solution. In the context of distributed application and protocols, the demonstration is organised into the environment of a lab. Obtaining an ad-hoc underlying network behaviour for the lab needs the help of the network emulator tools. The resulted behaviour can either emulate an operational real network technology or a specific condition to show particular efficiency of the tested software.

In this context, the first type of user could be an end-user, involved in a final application test or in the reception of specific software. The users need to be able to evaluate the use main user so that the network supervisor is able to plan the development of his/her network and to see the consequences of a faulty link or an increased capacity link on his/her network.

B.1.4 Requirements for Emulation Systems

The main issue regarding network emulation is to achieve experiments with protocols. An experiment is any process or study which results in the collection of data of which the outcome is unknown [186]. An experimenter is the actor conducting the experiment.

The term network emulation will be restricted to situations in which the user has control over some of the networking conditions under which the experiment takes place. This yields a new definition of an emulation experiment. An emulation experiment can be defined as deliberately imposing processing on the set of packets exchanged by distributed software in the interest of observing the resulting response. This differs from an observational study, which involves collecting and analysing the behaviour without changing existing conditions.

The various use cases of emulation previously introduced into the different stages of protocol design and developments lead to the following set of functional needs:

- Protocol or application testing: the emulation should be used to test a system under development, or an existing system with specific network conditions to reveal bugs, performances problems, deadlocks, or advantages. Therefore, modification can be performed on the product.
- Protocol performance analysis: the emulation should provide means for the system under development to be tested and compared to other systems, under specific network conditions or realistic network parameters to draw performance and comparison charts.
- Demonstration: the emulation platform will be used to show that the system under development is working under specific conditions and so be well-suited according to user need.

B.1.4.1 Functional Requirements

Considering the high level needs of network emulation, several requirements for network emulator can be defined. Note that all these requirements are not necessarily required for all uses of emulation:

- Controllability: the delay, loss, modification of packets must be achieved by using the emulation model that can totally be controlled. The model aim is to produce a specific set of impairments to either mimic real network architecture or to introduce a possible QoS scenario.

- Accuracy: The packets that have been computed to be dropped should be deleted. The remaining packets have to be delivered according to modification of the transmission delay calculated from the emulation model.
- Transparency: the emulation system should be used with a minimal (or no) modification into the tested protocol or application. Moreover, it should offer an interface classically used in the context to use the protocol, for example sockets for Internet applications.
- Flexibility: a large variety of methods should be provided to the experimenter to be able to compute and measure packet impairments. These methods should be adapted to either the development of an emulation model linked to real networking technologies, or to the development of a specific emulation model intended to stress a particular aspect of the tested protocol.
- Extensibility: The emulation should be able to facilitate further development of a new emulation model. The new model should be used to achieve particular requirements associated to the experimentation.
- Scalability: Scalability can be a very important factor, especially in the context of specific protocol testing such as high-speed or multicast protocols.
- Dynamics: The QoS in the communication area is evolving rapidly with various limitations, depending on underlying technology being used. In wireless environments (due to a variety of physical effect on the signal) and also in the context of wired networks (due to congestion, physical link failure, routing update, etc.), communication conditions may evolve. The emulation system should provide a way to test applications and protocols in evolving and different QoS conditions.
- Reproducibility: Experiment conditions that are developed to be carried out for testing or comparison between protocols should be reproducible to achieve fair comparisons and provide deterministic performance results.

B.1.4.2 Requirements on Packet Impairments

The basic action that the emulation system has to provide is to introduce QoS impairments on flows generated by the tested protocol. The impairments that can happen in a network are basically packet latency (delay), packet loss and packet modification.

The four main components of latency are propagation delay, transmission delay, processing delay and queuing delay (see Chapter 1).

The packet modification is mainly due to two possible sources. The transmission system can introduce bit errors due to fading or any signal level problem. This bit error could be transmitted to higher layers, if no error control mechanism is used in the lower layers. This is not a common case, but various studies at network and application level address this strategy and the gain that could produce in particular networking scenario. Another much more common type of packet modification is packet segmentation, which can happen in the network due to the necessary adaptation of transmission units in a heterogeneous networking environment. Furthermore, packet duplication is also a possible packet modification that can (rarely) happen in the network.

Packet loss is due to various situations in the network. Firstly, as stated in the previous paragraph, media transmission can produce bit errors that usually imply packet loss at a higher level, due to the loss of (checksum-based) detection techniques and associated protocol behaviour. Another important source of packet loss is due to buffer congestion. Buffer congestion can appear in many active network elements such as in a network router, but also in a link layer bridge or end-systems.

These impairments are the basic emulation actions that happen in a network. The impact of all these packet level actions is very important in higher protocols. Producing and controlling these impairments is a way to efficiently evaluate the functional and non-functional properties of these protocols and distributed applications. For testing purposes, these impairments can be produced to accurately stress a particular situation of loss, delay or packet modification. Similarly, for reproducing a global networking behaviour, these packet emulation actions are required.

B.1.5 Network Emulation System Approaches

Various approaches have been proposed to implement the general network emulation framework proposed in the previous section. In this section we will discuss each component in more detail.

B.1.5.1 Traffic Shapers

Once the experiment flows are constituted by the classifier, the emulation processor will add impairments to the packets; for example delay them, drop some of them and shape the flows to meet a given flow rate. A scheduler has to be implemented to manage the different existing queues.

Two approaches can be designed for the emulation controllers. The first one is when the emulation controller is implemented in a single computer (centralised approach) that represents the whole network. The second way to build an emulation processor is to use a distributed system (distributed approach) such as a grid, each computer having its own emulation processor and model, and representing a slice of the targeted network.

Centralised Approach

ONE [187] was a research project at Ohio University's Internetworking Research Group. It provides basic emulation of a network cloud between two interfaces, using a single computer running under the Solaris system. It does not manage several flows, so the user must provide a general model. It implements the following functionalities: bandwidth, buffer storage size, output queue size, RED algorithm, bit error rate and the delay.

Dummynet [188] integrates an emulation processor, working in the operating system kernel. It intercepts the packets between the network layer and the application layer and is able to delay them. It simulates/enforces queue and bandwidth limitations, delays, packet losses, and multipath effects by inserting two queues between the protocol layers, namely rq and pq. When a packet arrives, it is put into the rq queue that is bound and has its own policy (FIFO, RED for example). The packets are moved from rp to pq (which has a FIFO policy) at the given bandwidth. Once in pq, the packets are delayed by the amount of time specified. It can be used on user's workstations or on FreeBSD machines acting as routers or bridges. It is controlled by ipfw and some sysctrl commands. The user has to create pipes to classify the packets and then configure Dummynet to process the packets following these settings: bandwidth, queue size, delay and random packet loss. Dummynet also allows the user to create dynamic queues. All the packets matching the pipe rule will go in the same queue. With the ”mask” option, users can define more flows that will be split into different queues applying the same impairments. Each pipe is associated with one or more queues. Each queue has its own weight, size and discipline. A variant of Weighted Fair Queuing, named WFQ2+ is used to schedule the different queues of a single flow.

Although Dummynet was initially designed for studying TCP performance, NistNet [189] was designed from the beginning as a network emulator. It integrates an emulation processor and also works in the system kernel. It neither classifies the packets, nor the rate control. NistNet can apply the following effects on the flows: packet delay, both fixed and variable (jitter); packet reordering; packet loss, both random and congestion-dependent; packet duplication, and bandwidth limitations. The advantages of NistNet compared to Dummynet are mainly the statistical delay distribution it offers.

Netem [190] was developed on the basis of NistNet, which is not compatible with the 2.6.x version of Linux kernels. Netem is an emulation Linux queuing discipline, integrated into the Traffic Control (TC) module. It is only an emulation processor and does not include a classifier (i.e., see the various classifiers proposed by TC). Once TC has classified the packets into different classes, Netem impairs the flows. Finally, TC takes care of the queuing discipline and the transmission of the packets. Netem can provide the following effects on the packets: variable delay, choice of delay distribution, packet loss rate, packet duplication, packet re-ordering and flow differentiation. As it is based on TC, the user can choose all the other queuing disciplines and the shapers available for this tool.

EMPOWER [191] is an emulation project based on its own emulation processor. Each network node implements a Virtual Device Module (VDM). This module receiver is mapped to a network port and receives an egress flow, diverted from the IP operating system layer. Once in the VD module, the flow passes through six sub-modules, representing a different effect: MTU sub module, Delay sub-module, Bandwidth sub-module, Loss sub-module, Bit error sub-module and finally out of order sub-module. The flow goes through the sub-modules in the order previously mentioned. Once the last sub-module has passed, the flow is redirected to the underlying network port and transmitted. While in the bandwidth sub-module an on/off heterogeneous traffic can be injected in the VD to study their impact on the experimental flow.

Distributed Approach

Besides these solutions integrated in the kernel, another way to perform the emulation processing is to use a computer grid. A computer grid [192] is an association of a wide variety of geographically distributed computers, storage systems, databases and data sourcing interconnected via a network. Several applications like parallel calculus have been using grid computing, but it can also be used for emulation. These systems provide a way to implement a test bed as a configurable topology and a large amount of computers is provided. Some of the computers on the grid become an emulation processor and are connected to end-system computers with real links. Using low level emulator in certain nodes enables to produce bottleneck links. The whole grid behaviour can then represent a target network behaviour. These systems reproduce a real network and take into account side effects such as routing. We will present two different grid computing systems.

Emulab [193] is a grid computing oriented emulator. Emulab provides integrated access to three disparate experimental environments: simulated, emulated, and wide-area network test beds. Emulab unifies all three environments under a common user interface and integrates the three into a common framework. This framework provides abstractions, services, and name spaces common to all, such as allocation and naming of nodes and links. By mapping the abstractions into domain-specific mechanisms and internal names, Netbed masks much of the heterogeneity of the three approaches.

The Emulab emulation testbed consists of three sub-testbeds (nodes from each can be mixed and matched), each having a different research target:

- Mobile Wireless: Netbed has been deployed and opened to public external use, a small robotic testbed that will grow into a large mobile robotic wireless testbed. The small version (5 motes and 5 stargates on 5 robots are all remotely controllable, plus 25 static motes, many with attached sensor boards) is in an open area in our offices.
- Fixed 802.11 Wireless: Netbed's Fixed Wireless testbed consists of PC nodes that contain 802.11 a/b/g WiFi interfaces, and are scattered around a building at various locations on multiple floors. Experimenters can pick the nodes they want to use, and as with other fixed nodes, can replace the software that runs on the nodes all the way down to the operating system.
- Emulab Classic: a universally available time and space-shared network emulator, which achieves new levels of user-friendliness. Several hundred PCs in racks, combined with secure, user-friendly web-based tools, and driven by ns-compatible scripts or a Java GUI allow remote configuration and control of machines and links down to the hardware level. Packet loss, latency, bandwidth, and queue sizes can be user-defined. Even the OS disk contents can be fully and se-

curely replaced with custom images by any experimenter. Netbed can load up to a hundred disks in less than two minutes.

Grid 5000 [194] is a French project. It aims to interconnect geographically distributed clusters with high-speed links using Renater's network. Grid'Explorer is included in this project. This is a smaller grid designed to emulate network conditions. It will contain three components: 1000 nodes PC Cluster, an Experimental Condition Data Base and a tool set (emulators, simulators).

PlanetLab [195] is more a resource overlay network than a grid computing emulator. In this system, the Planetlab nodes are located across the world and linked using real Internet links. The nodes do not perform impairments on the packets arriving as they are interconnected using a real network. As PlanetLab implements real routers and real network protocols, this emulation platform is totally transparent to the user. The main critic made against PlanetLab is that it only reproduces a small slice of the Internet. It allows conducting experiments over computers linked by the research network (which is different from the commodity/commercial Internet).

The major drawbacks of these platforms are their low availability and their very high costs. Indeed, building a grid requires a large free space to store all the nodes and large funds to be able to buy the different computers. Once a grid has been set up, it is usually shared in order to use it at the maximum of its capacity. Reserving a grid is usually compulsory and it can sometimes take a long time to get a slot. The question of the grid administration has also to be solved. If the computers are geographically distributed, each site can administrate their own computers or a global administrator can be nominated.

B.1.5.2 Emulation Models

The emulation processors are a feasible way to process the experimented traffic. They must be controlled by the way of emulation models. Various levels of complexity in the emulation models are possible, depending on the aim of the whole experiment. There are three ways to obtain a network model: ad-hoc model (static or dynamic), trace-based model or by using a simulator.

Emulating QoS Parameters

An ad-hoc model is a set of parameters that are static during the whole experiment in this case, describing the network we want to emulate. It can be useful to reproduce artificial QoS. The parameters are the ones previously defined: delay, loss rate, BER, etc. With an ad-hoc model, the users design the network the way they want and get the QoS they need. With this type of ad hoc model, the network offers the same quality of service throughout the experiment. It can be useful to test all the possibilities of a product or to compare it to other already existing products. In order to set up these models, some GUIs are offered to the user by the emulation processor.

For example, NISTNet comes with a GUI while a PHP script has been developed to control Netem.

Ad-hoc models can also evolve according to external events, e.g., time. In reality, the QoS offered by a network is not the same during the night than during the day for example. Time-oriented models allow the user to define the network and to enable to evolve with time. For example, the delay will increase during high network utilisation. This can be useful to test a new product in different conditions or to schematically represent a general behaviour. The tester will be able to validate the product under several conditions and compare it to the other solutions. The Net Shaper [196] project uses time-oriented emulation. In this case a daemon is executed and is waiting for the new model to be applied to the emulation processor. The daemon was able to successfully receive and treat up to 1000 messages per second.

The events can also be driven by randomly generated events. In this case, the artificial QoS offered by the network would not be driven by time, but by an algorithm. The algorithm could represent, for example, a node failure randomly occurring or, in the case of a mobility impact study, the algorithm could modify the QoS offered, depending on where the user is located. The EMPOWER project [191] uses this kind of emulation in the wireless section. In this case, a VMN (Virtual Wireless Node) is added and is associated with an event table where randomly generated mobility events are listed. A time-stamp is also associated with the events. The incoming flows are impaired according to this event table.

A last event driven ad hoc model is the script driven model [197] [198]. In this case, a script describes what the emulator processor should do (e.g., impair the flow differently, transmit another packet, reply to this packet, etc.) when it receives a packet. It is useful to study experimental protocols and determine their reactions under several conditions.

Virtual Nodes Approach

In this approach, the global network behaviour is produced by virtually reproducing the network topology and components. All nodes constituting the target network to be emulated are implemented either on a single centralised system or distributed on various distinct systems usually connected by high speed networks. Virtual links are used to connect these nodes according to the topology of the target network. Real protocols such as IP or routing protocols can also be implemented in the virtual node system. This approach can be implemented in a system (several virtual nodes coexist in the centralised system) or over a distributed system such as a grid. Of course, in this type of architecture, the classical strategy to produce realistic behaviour is to introduce real traffic into the emulated network to produce congestion, etc.

IMUNES [199] is an example of the centralised virtual node approach and proposes a methodology for emulating computer networks by using a general purpose OS Kernel partitioned into multiple lightweight virtual nodes, which can be connected via kernel-level links to form arbitrary complex network topologies.

IMUNES provides each virtual node with a stack that is independent of the entire standard network, thus enabling highly realistic and detailed emulation of network routers. It also enables user-level applications to run within the virtual nodes.

At the user level, IMUNES proposes a very convenient interface enabling the target emulated network to be defined, defining the virtual nodes and their software and links as well as the impairments parameters.

Another comparable approach is the Entrapid protocol development environment [200] that introduces a model of multiple virtualised networking kernels and presents variants of the standards BSD network stack in multiple instances that are running as threads in specialised user process. Other approaches that follow this approach are the Alpine simulator [201] project and Virtual Routers [202].

This approach is often considered the only means to achieve realistic emulation of complex network topology. It enables the target network to be specified accurately and realistic behaviour—as a direct effect of real traffic and protocol implementations running in the system—to be produced. Nevertheless, the major problem of this approach is scalability. How can one implement a core network router in a single machine? What if a realistic network contains several dozens of such network elements? How can one manage the number of necessary flows to produce realistic conditions in a centralised manner? These questions are very difficult to answer, in particular in the context of a total centralisation, but are real problems in distributed systems like grids. Moreover, most of the emulation experiments do not require such an approach to be implemented. What is really needed by most protocol experiments, is the notion of channels interconnecting the various protocol entities that are under testing with either a realistic or a specific behaviour.

Trace Based Approach

The trace based approach is another way to obtain realistic behaviour according to a given network infrastructure. By using different probes on the real network, or even into a modelled network (e.g., with a discrete event simulator) it is possible to get the different parameters aiming to reproduce the network behaviour.

The trace based approach consists of recording the performance of a real (or a simulated) network, and then use these traces to drive the emulation processor impairments. In Virtual Routers [203], an appropriate methodology for the trace based approach is provided. It consists of three complementary phases:

- Collection: in this phase, an experimenter with an instrumented host takes measurements. In the case of a wireless network, the experimenter can be mobile and traverse a path. During the collection, packets from a known workload are generated. The mobile host records observations of these packets. By performing multiple traversals of the same path, one can obtain a trace family that captures the network quality variation on that path.
- Distillation: The distillation phase transforms a collected trace into a form suitable for the next phase. For each time instant, the distillation examines the performance of the known workload and produces a set of parameters for a simple

network performance model. By composing these, distillation produces a concise, time-varying description of the network performance.
- Modulation: in the modulation phase, the system under test is subject to network impairment driven by the previous network performance model.

The parameters associated to the end-to-end channel are for example delay, loss rate, packet reordering or packets duplicated [203]. The user must have probes on the end-user stations that are used to record the dates the packets arrive or leave the station. Once this is done, the results are transmitted to a controller that evaluates the delay and the mean loss rate, given the network model. This enables the user to obtain a dynamic network profile. Depending on the limitations of the trace (over a month, or over an hour) we obtain general or specific network conditions. The trace based approach has one major drawback: it can hardly reproduce all the conditions that a network can encounter. The trace based approach is not a panacea. A single trace can only capture a snapshot of the varying performance along a particular path. Moreover, the traces cannot offer precise reproducible results because network processes are non deterministic, and the same situation at another time could have produced further impairments.

Simulation Based Approach

Using a simulator, like a discrete event simulator, is a very common way to conduct experiments in networking (see Chapter A). The key characteristics of Discrete Event Simulations are:

- Events occur at specific moments in time.
- Polling is done to find the time of the next event.
- Time does not increment by fixed amounts. Time jumps to the time of the next event.

An event occurs anytime the state of the simulation needs to be updated. This happens for example when a packet is created, when it is sent from one node to another, or even during the processing of a packet in a node (if required).

Event driven simulators provide a very convenient and very useful way to model networks. The idea here is to provide the ability to use the simulator capacity to drive the emulation processor. This cooperation is achieved at a price of having to undertake several modifications on the original tool. The simulator has to work in real-time and needs to capture and generate real data packets. Fig. B.3 depicts the general model of discrete event simulator aided emulation.

NS [204] is a widely used discrete event simulator (see Section A.4). The international networking research community has greatly contributed to the models and tools used in this simulator. An extension to this tool, nse [205], allows integration between modelling capacity with the impairments implemented in real-time and applied to live traffic.

Of course, various modifications have been designed to enable such utilisation. First, the event driven scheduler has to be replaced with a real-time scheduler. In

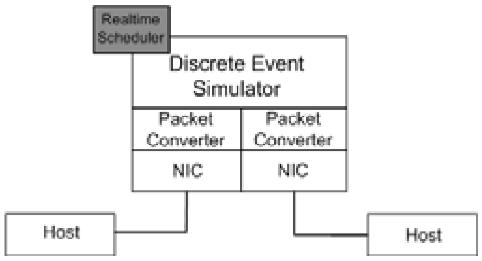

Fig. B.3 Using discrete event simulator to provide emulation behavior.

the case of emulation, the simulator does not have to jump onto the next event once the last event has been processed. It is crucial to process the events synchronously in real-time. This real-time synchronisation is an important limitation of the tools capacity. For example, if too many events occur at a given time, the simulator still has to process them at the right time, not before and not after. Then, the simulator can come into a ”livelocked” state, if a packet arrived while it delays the last event. Obtaining a really accurate real-time clock is merely impossible and that causes the simulator to misbehave under heavy traffic conditions. Another drawback is that if a large delay is applied to the packets, large memory systems are required to store all the events to be treated.

Another task to perform to get NS working is to interface it with real traffic. Indeed, to ”route” packets through the simulator, NS uses its own representation of the network, totally different from the IPv4 or IPv6 world. When a packet arrives, two possibilities can be considered: the real IP address can be mapped to the NS address or NS could be modified to understand IP addresses. The first possibility is implemented. Finally, NS also has to be modified to the process of capturing and injecting real data and the packets going through the network. For this task, NS provides a network object that can understand UDP/IP, raw IP and frame level data. To capture UDP/IP and raw IP packets it uses the standard socket API while using the Berkeley Packet Filter to intercept the frame level packets.

Nse can perform emulation in two different modes: opaque and protocol. In the opaque mode, the packets are passed through the computer without being interpreted. NS implements an emulation processor allowing the real packets to be dropped, delayed, re-ordered or duplicated depending on the simulation results. Opaque mode is useful in evaluating the behaviour of real-world implementations not interfering with the model.

In the protocol mode, packets are interpreted by the ns model. The real packets are translated into simulation events and take part in the simulation just like any other internal event. The simulated protocols can then react depending on the real packet. An inverse function allows the simulator to convert internal packets into real ones and inject them into the network. The protocol mode can be used for end-to-end application testing, protocol and conformance testing. Since the protocol mode includes a protocol agent understanding the headers, it enables the users to drop packets depending on header flags.

This approach is theoretically very powerful with respect to the modelling possibilities it may offer. Nevertheless, the major drawback of this approach is the scalability problem requiring extensive system memory and CPU speed to handle the potentially large number of packets on the network. Other approaches using simulation for emulation have been developed to improve the scalability of the virtual network. For example, in IP-TNE (Internet Protocol Traffic and Network Emulator) [206] the use of parallel discrete event simulation and simulation abstraction such as fluid simulation are addressed to improve the simulation. Various models can be applied in order to configure the emulation processors. Each model has its advantages and drawbacks, depending on the experiment the user wants to conduct. In all cases, getting an exact representation of the reality is a difficult task to implement.

Active Emulation Approach

This section outlines the concept of active network emulation. We will see how active emulation can be a realistic way to emulate complex network behaviour, and an application of this concept will be presented in the next chapter to emulate a satellite link.

Some networks produce network dynamics that may require complex models and mechanism to be emulated. The complexity of this type of emulation often comes from the highly dynamic behaviour of the protocols used in such networks. Protocols react depending on internal factors like their own mechanisms and external factors (such as the traffic crossing the network). For instance, satellite networks, in particular access schemes such as DAMA, propose three main different types of traffic assignment techniques that may be combined. Some of them depend on the traffic characteristics, while others do not. Since the behaviour of this access scheme is not predictable in advance, dynamic configurations of the emulation system are required. The only way to have sufficient realistic emulation behaviour is to react in real time on the emulation model, according to various factors and including the experimental traffic. In this sense, emulation needs to be active: the traffic will modify the configuration of the emulation and consequently the resulting modifications will have an effect on the traffic crossing the emulator. Considering this emulation problem, several techniques might be used to emulate a satellite link.

In the context of low-level protocol developments, satellite emulation systems can be based mostly on real protocol implementations (e.g., DAMA, adapted routing protocols, etc.), using a wire emulation model to avoid the use of a real air interface. The protocol's complete behaviour and signalling is then actually implemented while only the physical link is emulated, introducing geostationary end-to-end delay, and possibly loss models. Such systems are often very accurate in the emulation service they provide, but are also very complex and expensive to develop and maintain when protocols evolve. In the context of end-to-end communication, protocol experimentations do not need such low level emulation.

The active emulation approach is an alternative [207]: Instead of reproducing access to the satellite link according to a heavy implementation of low-level protocol behaviour, only resulting effects on data transfer are emulated by the way of a proper emulation model. This results in a simple and practical implementation that can be thus combined with more complex emulation scenarios (e.g., for emulating a wider network including a satellite link) or operational networks. To achieve this goal, the emulation model must react in real-time to various external events such as time or processed traffic, leading to the concept of active emulation. The advantage compared to the first type of emulation is that it provides an ”in-a-box” solution, integrating an emulation processor and a potentially complex and realistic emulation model (not limited to satellite links) into an easy-to-deploy system.

B.1.5.3 Implementation

The emulation platform can be implemented in different layers, ranging from the hardware layer up to the application layer.

User Space Implementation

Random events and programming facilities in user space have been the reason for many emulators being developed at this level. The major challenge is that even a more present, but user-friendly GUI, and a widely tweaked application need to be developed to solve the problems. In these architectures, the software has to intercept the packets in the network stack to then process and re-inject them. It can either trick the operating system into making it believe that it is in the network stack, or either intercept the packets and inject them using its own sockets, enabling to add further services on the classical sockets. Some projects dynamically configure the emulator running in the operating system stack. Although the packets are not captured, the modules in the network stack are constantly modified.

User space offers lots of services, programming tools and libraries that facilitate building complex network emulation models. One of the major requirements of such implementations is to have access to packets that usually cross the system where the user space emulator is implemented. Various possibilities exist to capture those packets, such as Raw IP socket, Libpcap and libnet or Divert socket.

The main advantage of the user level implementation is the flexibility of services and development that it offers in comparison to kernel space. The two main drawbacks of user level implementations are:

- At least one packet copy exists between the system kernel and the user space, affecting the performance of the processed streams;
- User space processes can be interrupted. It basically means that a higher priority task can be scheduled instead of the emulation task, with all the timing and processing.

NS Emulation including its Emulab front-end and the Ohio-Network-Emulator ONE is an example of emulators implemented in user space.

Kernel Implementation

Network emulation is time critical. As stated in the general requirements of network emulation, the delay impairment must be achieved as precisely as possible and user level implementations might fail on this real-time task. A possible alternative to implement emulation processor is to do it in the operating system kernel to optimise the performance. In this context, various emulators such as dummynet or NISTnet are implemented between the Ethernet and IP layers.

Packet interception is completely hidden from user space programs and any user space software using the standard UNIX communication interfaces (sockets, raw sockets) will be affected by the specified network properties. The overhead introduced by the additional layer a packet has to pass is crucial, because packet data has not to been modified or copied to different memory areas (unlike with user space approaches). Communication through interfaces with NETShaper is required.

The kernel based-solutions offer a powerful set of possibilities that are cheap to develop and do not introduce much overhead in the packets. The major drawbacks of these solutions is that they are limited to the precision of the operating system clock and that if the operating system is highly loaded, some clock ticks can be missed, impacting the platform performance. Furthermore, development of applications within the operating kernel is harder than on the application layer. The developer has less tools and functions to program, has to be careful with the memory management and the stability of the software. It is highly probable that if the module crashes, the kernel crashes too.

Most of emulation systems like dummynet or NISTNET are implemented in the kernel to provide high performance. The implementation of the emulator on a real-time operating system might be a potential solution to the problem, which needs further investigations.

Hardware Implementation

An emulation implemented with hardware (e.g., FPGAs) can possibly perform better. The emulator needs to be set up based on the parameters given by the user. It processes the flow with limited software overhead. This approach has the advantage of minimising the processing time of each packet, but it is also less flexible, harder to design and more expensive to develop than using software included in the operating system network stack. The emulator can also be implemented with dedicated hardware like a network processor, but the software is harder to develop than under regular X86 architectures, even if the flows are treated more efficiently.

B.2 Case Study: Emulation of QoS-oriented Satellite Communication

B.2.1 Introduction

Among the large set of network types considered into the EuQoS project, a geostationary satellite link is particularly interesting for experiments, due to the particular QoS services it offers. The satellite link targeted is based on the widely spread DVB-S standard for the Forward link (from the Hub Station toward the Satellite Terminal) and DVB-RCS standard for the Return link (from the Satellite Terminal toward the Hub Station) [208]. This satellite access network has to be provided in order to make the corresponding experimentations when a satellite link is part of the end-to-end communication. Due to the high cost of satellite resources, experimentations involving this particular access network will be conducted on an emulated satellite link. In order to develop this emulation platform, the characteristics of the satellite link will be firstly presented.

The main issue in the satellite communication context, and more particularly on the satellite return links, which will be the particular subject of this use case, is to make an efficient use of the precious transmission resources, scarce and costly. Recent techniques based on dynamic bandwidth assignment, enable a high efficiency of the return link usage. Emerging protocols, such as DAMA (Demand Assignment Multiple Access) integrates a combination of these existing protocols in order to both ensure a high utilisation of the return link resources and to offer QoS oriented capacity assignment types. This access scheme is targeted for the satellite access network experimentation instance presented into this case-study.

B.2.2 DVB Satellite Communications

DVB-S and DVB-RCS are standards used to carry out IP-based applications over geostationary satellite: **DVB-S** (Digital Video Broadcasting - Satellite) is used to transport data over the forward link (from the gateway earth station to the numerous satellite terminals). **DVB-RCS** (DVB – Return Channel System via Satellite) is used to transport data over the return link and specify the access scheme to the return link (from the satellite terminal to the gateway earth station). This standard enables to share efficiently resources between a great numbers of Satellite Terminals (ST) accessing the return link. We will now see in detail how access and resource allocation is managed by the protocol and how we can take advantage of it to introduce QoS differentiation on this kind of link.

The satellite user terminal receives a standard DVB-S transmission generated by the satellite hub station (the gateway). Packet data may be sent over this forward link in the usual way: DVB-S [209] defines several ways to encapsulate data packets into an ISO MPEG-2 Transport Stream [210], but the common practice for IP datagrams encapsulation is to use the DSM-CC sections through an adaptation layer protocol, named Multi-Protocol Encapsulation (MPE) [211]. DVB-RCS standard [208] is associated to the use of AAL5/ATM, but can also use MPEG2-TS Stream.

To get an idea of the availability and requirements of QoS in the targeted satellite system, here is a description of how the DVB-RCS system works:

A Return Channel Satellite Terminal (RCST), receives general network information from the DVB-RCS Network Control Centre (NCC), sent over the forward link, to get control and timing messages.

All data transmissions by the RCST over the return link are controlled by the NCC. This dynamic resource admission control permits to assure an optimal use of the costly resources of the satellite.

Dynamic resource control consists in assignment of resources (slots) to STs based on their requests to the NCC and limit values negotiated during connection establishment. The assignments are conditioned by the availability of resources within defined return channels. The assignment is the responsibility of the MAC Scheduler (in the NCC), which implements a Demand-Assignment Multiple Access (DAMA) protocol.

The uplink scheduling consists of processes taking place in the scheduler and in STs: First, STs calculate capacity request required for the current traffic and send it to the NCC. Then NCC calculates and sends the overall assignment to every ST of the satellite system taking into account current load of the system as well as requests and limitations of specific ST. Finally, the capacity is distributed within terminals to end-users and their applications (depending on ST MAC queuing architecture and service discipline).

The Service Level Agreement (SLA) between the terminal and the hub specifies guarantees on different classes of access to the Return Link of the satellite. These classes are defined in the DVB-RCS standard as capacity allocation of a different type [208].

The DAMA implementation of DVB-RCS uses a combination of static and dynamic allocation techniques in order to ensure a set of QoS guarantees as well as high bandwidth efficiency. The return link scheduler supports three main capacity assignment types to reach this objective, described as follows:

- **Fixed rate** (Continuous Rate Assignment – **CRA**).The CRA assignment type is a guaranteed rate capacity, fully provided for the duration of the connection between a ST and a Satellite System, without any DAMA request. The delay associated to this capacity assignment is fairly constant and reduced to the propagation delay of the satellite link.
- **Variable Rate** (Rate Based Dynamic Capacity – **RBDC**). This traffic assignment is based on requests depending on the average rate of incoming data on the ST. This assignment type can be guaranteed (up to RBDCmax ceiling rate) or not, but always on demand. The rate assignment is valid for a certain period of time: after the timer has expired, capacity is not assigned anymore except if a request was done in the meantime. A sustained traffic will be doing periodical requests, thus avoiding timers expiration. In this case, the delay associated to this capacity assignment, after the initial requests, will be equal to the propagation delay.

- **Best effort** (Volume Based Dynamic Capacity – **VBDC**). This traffic assignment is based on requests indicating the volume of data in the ST buffers. The capacity is assigned when available in response to a request, without any guarantee on assignment. The delay for traffic using this capacity assignment type can be long (if capacity is not available) and may vary considerably. A guaranteed VBDC capacity can also be defined by setting a minimum value for VBDC (MinVBDC) per ST. VBDC capacity up to MinVBDC will be granted (when requested), in every super-frame.

The return access scheme of the satellite is able to provide different types of service. However, QoS differentiation can not be done without architectural solutions at upper layers. The next section presents a brief overview of these solutions as well as the QoS-oriented architecture targeted for the emulation.

B.2.3 QoS Support for Satellite Network Systems

In the satellite networking context, the interaction between the IP Layer where the QoS might be set, and the lower layers where the traffic is finally prioritised, is of major importance. QoS techniques and architectures for satellite networks have been widely studied in the literature and the standardisation of these QoS architectures are in progress.[212] [213] propose to use DiffServ architecture [14] on both forward and return link. This architecture is well adapted to the return link due to the different classes-of-service of the DVB-RCS capacity allocation. The satellite system, in this study, is assumed to be an access network to the Internet for end-users. Thus, as a boundary node, the ST is the most important component regarding QoS support on the Return Link. It has to implement traffic conditioning/policing functions, in addition to packet classification and per hop forwarding/scheduling according to a packet's Class-of-Service, as illustrated in Fig. B.4.

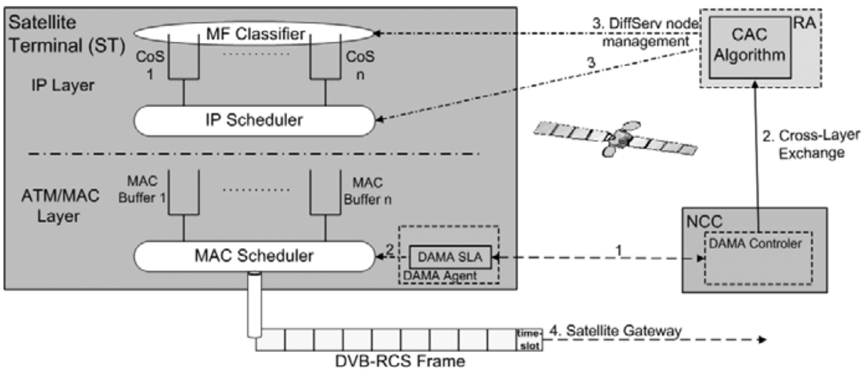

Fig. B.4 QoS-oriented Architecture of the Satellite Terminal in EuQoS

The proposed mapping and admission control based on these recommendations are as follows:

The end-to-end QoS architecture deployed in EuQoS system, integrates DiffServ and includes signalling mechanisms for admission control as well as resource pre-reservation. In order to meet the DiffServ forwarding requirements, the IP classes of service need to be appropriately mapped into MAC QoS classes and then into DAMA capacity categories supported by the Scheduler. Here we consider that **RT** traffic is directed to the highest priority MAC buffer DVB-RT using CRA capacity. **NRT** traffic is redirected to a medium priority MAC buffer DVB-VR using RBDC capacity. **Elastic** traffic is redirected to the lowest priority MAC buffer DVB-JT using VBDC and the remaining capacity. The admission control is done by the Resource Manager (RM) and Resource Allocator (RA) depending on the available satellite resources on the return link. This information is passed from the Network Control Center (NCC) of the Satellite System to the RA. A Service Level Agreement (SLA) is passed at logon between ST and the satellite system's NCC. The bandwidth guaranteed for high priority classes (CRA, RBDC) in this SLA, are generally restricted due to their cost. Thus, to avoid the waste of high priority capacity, admission control is based on the remaining satellite resources and limitations per end-user. If bandwidth is available for a specific IP CoS (RT or NRT) in relation with remaining satellite resources in the corresponding DAMA class, the flow is admitted, if the user is under its contract limitation. No per-flow admission control is done for the elastic CoS type, but its global rate is limited to the remaining bandwidth not used by high priority traffic. This ensures full resource utilisation while limiting congestion in the ST. This QoS Architecture including differentiated services and admission control enables a flow to use a satellite access class on the return link, without any interference with concurrent traffic. Thus, a prioritised flow is able to use CRA or RBDC access class, and will not be delayed by Best Effort traffic that would rather use VBDC capacity when available. However, the pre-reservation of resources realised by the QoS architecture is different from the actual allocation done by the satellite system with internal requests. Indeed, reservation could be pre-reserved on the control plane, but the connection could finally fail. Thus, immediate allocation of resource in a satellite environment is not feasible due to its cost.

B.2.4 Emulation of a DVB-S, DVB-RCS Satellite System

Emulation platforms are a classical way to achieve protocol experiments, particularly in the expensive and complex satellite environment. In the context of low level protocol developments, satellite emulation systems can be based on real protocol implementations (e.g., DAMA, adapted routing protocols, etc.), using the wire emulation model to avoid the use of real air interface. The complete protocol behaviour and signalling is then implemented while only the physical link is emulated, introducing geostationary end-to-end delay, and possibly loss models. Such systems are often very accurate in the emulation service they provide, but they are also very complex and expensive to develop and maintain when protocols evolve. In the context of end-to-end communication, protocol experiments do not need such low level emulation.

In the active emulation approach proposed here, instead of complex implementations of link layer access (in this case the satellite link) according to a protocol, only resulting effects on data transfer are emulated by a proper emulation model. This leads to a simple and powerful implementation that can be thus combined with more complex emulation scenarios (e.g., for emulating a wider network including a satellite link) or operational networks. To achieve this goal, the emulation model must react in real-time to various external events such as time or processed traffic, leading to the concept of active emulation. The advantage compared to the first type of emulation is that it provides an "in-a-box" solution integrating an emulation processor and potentially complex and realistic emulation model (not limited to satellite links) into an easy-to-deploy system.

B.2.4.1 Integration of an Emulated Satellite Link and the EuQoS System

Fig. B.5 presents a simplified scenario of the target satellite system previously described, integrated into the EuQoS architecture.

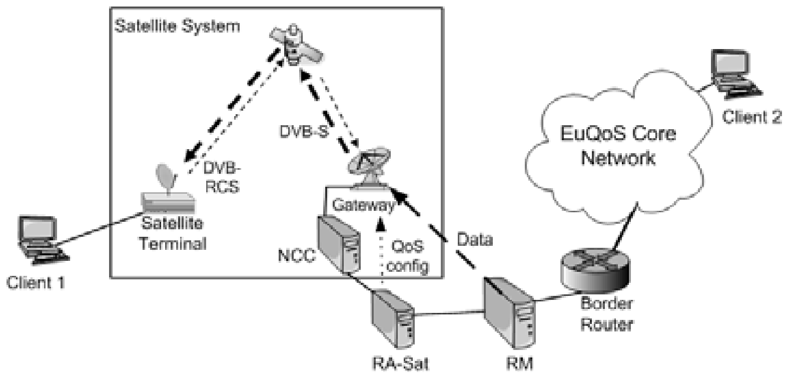

Fig. B.5 Simplified Target Satellite System

The satellite system presented in Fig. B.5 is emulated by a single physical component, the Emulation System. The Emulation System is managed by an emulation controller, configured by an experimenter through predefined scenarios. This emulation controller also integrates the Resource Allocator specific to the satellite system implementation that also influences the Emulation System (configuration of overall resources and admitted flows). This RA-Sat manages the technology-dependent QoS provisioning specified by the RA-Controller through the EuQoS standard interface. The RA-Sat provides information to the RA-Controller on the current status of satellite system utilisation.

The Resource Manager of the satellite system is running on the same machine as the RA-Controller and manages Admission Control to the Satellite System. Two sets of modules are accessing the Emulation System:

- The first one represents the satellite Resource Allocator modules defined in EuQoS namely:
  1. The Connection Admission Control algorithm specific to the Satellite Link
  2. The Underlying Network Configuration Module, that need to configure the satellite link (here the satellite emulator) upon traffic admission or release.
- The second one corresponds to the scenario-based emulation control. Scenarios are defined by experimenter (using XML files) in order to specify and emulate concurrent cross-traffic on the satellite terminal as well as overall load of the satellite system.

To access simultaneously to the satellite link emulation, these two sets of modules use a common class instantiated for each satellite access classes. Then, these objects send messages to the Emulation Control module.

The Impairment System is able to apply impairments to the traffic concerning three types of parameters:

1. the delay experienced,
2. the bandwidth limitation,
3. the loss model.

Messages sent by the satellite RA and scenario control modules through access classes objects are translated into control messages sent to the impairment system. These control messages have an impact on the bandwidth allocated to the traffic and the packet loss model experienced by the traffic crossing the emulator. Concerning the delays applied, messages do not modify them; the delay variations reflect the intrinsic behaviour of the access scheme and are not influenced by external events other than real data (like for instance admission of flows or cross-traffic emulation), in the case of satellite emulation.

The behaviour of each Class-of-Service of the EuQoS System in the satellite context is pre-configured in this Emulation Control module. The Classes-of-Service behaviour for the return link is directly linked with the Satellite Access classes considering the mapping proposed.

Emulation control and impairment system can produce basic behaviour that can be composed to produce more complex behaviour including interaction with the traffic crossing the emulator leading to the "active" aspect of this architecture. The composition of such basic impairment modules may correspond to a specific behaviour of a network, used to evaluate a protocol in this context, or to produce a target technology, such as the QoS enabled satellite link. The emulation control is also in charge of managing information on the traffic. The impairment system finally applies rules to interfaces in order to configure the impairments on the traffic as needed. It is also in charge to provide in real time required information about the traffic to the emulation control, such as sequence number of packets, packets' size or any kind of useful information for emulation control modules in order for them to take some decisions on the evolution of the emulation and thus to apply appropriate rules to the ongoing traffic. The concept of active emulation corresponds to the dynamic configuration of the impairment system depending on information gathered in real-time from the data crossing the system; on the opposite, passive emulation could refer to emulation models where scenarios are defined in advance, independently of the traffic crossing the impairment system. A traffic shaper is then in charge of applying constraints on traffic crossing the emulator in accordance with the current configuration. A detailed description of this active impairment framework and emulation control is described in the next section.

B.2.4.2 Active Emulation of DVB-RCS Access Scheme

As presented earlier, three main IP Classes-of-Service are used to provide different levels of QoS in the EuQoS system. A distinct emulation of these three classes needs to be achieved for the satellite system, on forward and on return links. We will detail how these access classes are emulated using the active emulation concept after a description of the impairment framework and emulation control.

Impairment Framework and Emulation Control.

The impairment system that will be used to produce the QoS-enabled satellite behaviour uses a framework based on the experimentation channel (EChannel). The EChannel component offers a target QoS to the System under Test (SuT). The EChannel is defined as a data path providing to the SuT particular QoS impairments. The actual EChannel QoS will be here associated with the DVB technology.

The Experiment Channel, illustrated in Fig. B.6, intends to produce the final target behaviour in terms of QoS for the experimentation. This resulting behaviour can implement a very simple behaviour such as constant end-to-end delay or a more complicated one such as the behaviour of an end-to-end path constituted by a various underlying network technologies. To allow the implementation of such an arbitrary complexity, the EChannel is built by the composition of Experimentation Nodes (ENodes) having a programmable action on the traffic. Each ENode is an active component that offers the necessary communication ports to achieve the internal communication of experiment packets. The nodes are individually parameterised using an additional communication port (pConf). Finally, a specific spying port called pSpy may be used to give information about the ongoing processed traffic to an external management module. The communication and spying ports (pConf and pSpy) enable active emulation. InputTap and OutputTap are input and output interfaces ensuring the packet capture and re-injection into the physical network.

As illustrated in Fig. B.7, the experimentation node is divided into two main parts to differentiate the actual impairments to achieve and the controlling process: the emulation processor is impairing the packets while the emulation model is deciding

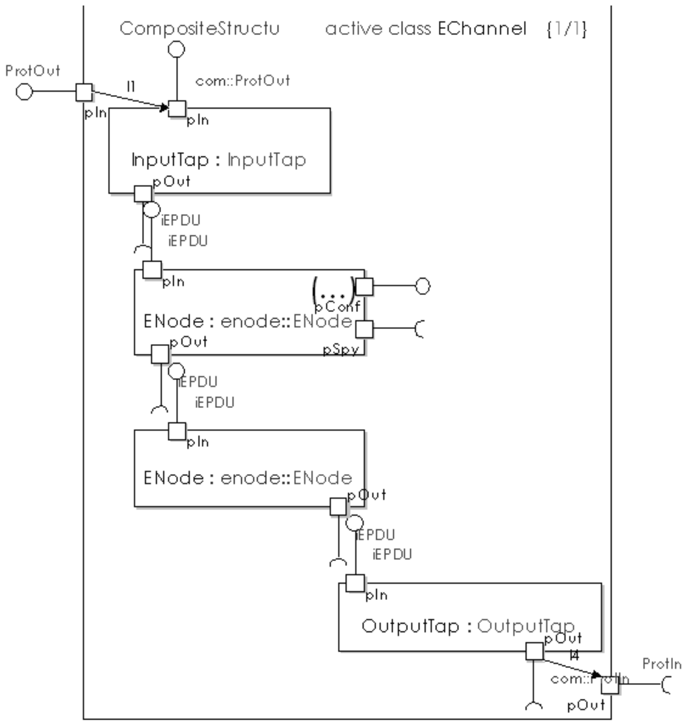

Fig. B.6 Experimentation Channel is a way to produce a target behaviour.

how the packet will be processed, having access to various information (i.e., length of packet, capture time, internal packet fields, external ports, etc.). When a packet enters into the ENode, the processor asks the model what to do with it. The model can be arbitrary complex, but needs to reply to the processor which really processes the packet. The various actions in the processor are delaying, dropping, modifying, etc.

The experimentation model provides an abstract representation aiming at specifying actions to be taken on packets. Experimentation channel processing can be defined either statically or can evolve dynamically during the experiment.

In complex models, various external events can drive the processing of packets like algorithm (e.g., a random function), time (e.g., a leaky bucket implementing a traffic shaper producing a bandwidth limitation). In the context of active experimentation, measurements on the processed traffic or data contained into the packet themselves can be performed. Mixing those various possibilities allows implementing arbitrary complex per-flow behaviour.

As previously introduced, two main types of models are defined, namely passive and active models. Passive models act on packet events considering arrival time and packet length. Examples of classical passive models implemented are e.g., delay,

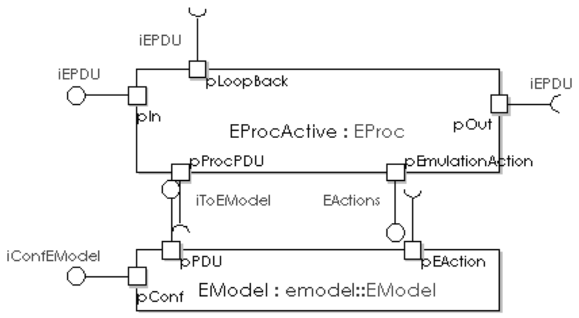

Fig. B.7 Experimentation Node is composed of a Experimentation Processor and Experiment Model

jitter, packet loss, packet re-ordering, packet alteration, etc. Active models can react on any stimuli in addition to the packet event itself. Those stimuli can be time-driven or packet-driven based on the value of the data contained in the emulation packet, or any other signals coming from real world like a geographic position information, or even signals coming from the state machines of the SuT.

All these models (both active and passive) can be composed in order to obtain an arbitrary complex behaviour. ENodes are proposed as an extensible library intended to provide various types of experimentation processing. The end-user can then compose the channel depending on experiment objectives.

B.2.4.3 Emulation Details of the Satellite System in EuQoS Architecture

We will now detail the way the satellite link and each Class-of-Service is emulated using the active emulation concept. Emulation of Forward Link access is presented first. Then emulation of various access classes of return link as well as combination of these classes is presented.

Emulation of the Satellite Forward Link

The access to the satellite link on the forward link is centralised on the gateway. This implies that there is no specific protocol to access the channel, but just a classical allocation of resource to an aggregate and an encapsulation in the MPEG2/DVB-S frame. The encapsulation is taken into account by the control modules to set the total bandwidth allocated for each Class-of-Service and to update this bandwidth at the time the flows are admitted.

The Forward link is proposed as a simple experiment channel integrating a constant delay, a throughput shaper and a loss rate ENodes. For each Class-of-Service the same delay, corresponding to the link crossing, is experienced on the forward link and thus needs to be emulated by the impairment system. In lower priority classes, additional delay might be experienced due to larger buffers. The delay experienced to cross a geostationary satellite link is around 250 ms, this is the delay applied to packets crossing the Forward Link in **RT**, **NRT** or **Elastic Classes**.

The difference between these classes is the buffers sizes that are assigned, implying different delay and losses for the traffic:

For **RT class**, the buffer size is reduced to the minimum, because the traffic of this class does not admit additional delay. Flows are admitted if the required bandwidth is available in the class. This requires that the input traffic rate is lower or equal to the output traffic rate.

For **NRT class** the buffer size is reduced, but not as much as for the RT class, because the traffic admitted in this class is more prone to fluctuations, and thus the buffer needs to absorb potential bursts. Besides, the delay is not the main issue of this Class-of-Service.

The **Elastic Class** is able to take advantage of resources unused by the other classes and is tolerant to delay. Thus, the buffers need to have rather large dimensions in order to efficiently use the potential resource unused by other classes.

The bandwidth impairment for each Class-of-Service depends on a static consideration: the agreement considered between the satellite terminal and the satellite system's NCC (including guaranteed resource for each access class) as well as on a dynamic consideration: the flows admitted in the class. A simple configuration through high-level modules is possible in order to set this bandwidth limitation on the throughput ENode. The applied loss model is the same for all Classes-of-Service and depends on the loss model defined in the scenario (based on weather conditions).

Emulation of the Satellite Return Link

The emulation of the return link is more complex than for the forward link. Several Satellite Terminals access the link simultaneously. Thus, a more complex access scheme has been defined to share efficiently the resource between them. The emulation of on-demand capacity allocation types requires the usage of active emulation.

Depending on the traffic assignment type to emulate, predefined QoS parameters, such as delay and throughput, are initially set by control modules, in order to emulate propagation delay or bandwidth limitations. Some QoS parameters are then set in real-time in order to emulate signalling process, cross-traffic and satellite load, and take into account for admission of new flows. The signalling protocol is not implemented, but only its resulting effect on the traffic.

The emulation control is based on the experimentation channel defined in the impairment framework to perform impairment decisions. The spying and communication ports of emulation nodes enable to communicate information about ongoing processed traffic and to apply real-time impairment decisions taken.

The experiment channel implementing the return link uses a more complex Enodes composition than the forward link, especially for on-demand access classes. An active ENode has to dynamically compute the bandwidth limitation, delay and packet loss rate values to apply to the set of ENodes, which actually impair the flows according to the initial configuration of the satellite link emulation and the information about ongoing network traffic. Depending on the initial configuration corresponding to the predefined type of contract, several types of traffic assignment classes will be available and accordingly instantiated in the emulator. Depending on the traffic assignment types and the ongoing traffic, the active ENode will initially set some values for the parameters such as delay and throughput, and will take the decision to modify these values in real-time. This will be done in order to produce the delay introduced by the signalling process, depending on the resources delivered to a particular Satellite Terminal.

The three main IP Classes-of-Service of the EuQoS framework are mapped to the three access classes of DVB-RCS as follows:

- **RT Class** is mapped to the CRA allocation type. This access class offers a guaranteed bandwidth permanently allocated (without signalling) during the entire connection. Delay for the traffic using this class is then reduced to propagation delay.
- The **NRT class** is mapped to the RBDC allocation type. The capacity allocated to this class is on demand and requested based on the data rate entering the ST buffers.
- The **Elastic class** is mapped to the VBDC allocation type. The capacity allocated to this access class is on demand, and requested based on the data volume in the ST buffer.

The emulation of the RT Class is similar as on the forward link and uses a simple experimentation channel composed of three ENodes:

- The first ENode emulates a constant delay of 250 ms that is applied to packets, according to the propagation delay.
- The second ENode is a throughput shaper. The bandwidth impairment depends on the SLA passed between the ST and the Satellite System's NCC as well as the admission of real flows by the Resource Allocator or emulated ones.
- The third ENode emulates the loss rate. The loss model depends on the defined scenario (based on weather conditions).

The experimentation channel used for NRT and Elastic classes are based on the model of active emulation. Active emulation is required for on-demand traffic, because the behaviour of the experimentation channel depends on the traffic injected by the SuT. In particular, the signalling part of this protocol has a non-negligible effect on the delay and bandwidth experienced by the traffic reaching a satellite return link. Fig. B.8 shows the used experimentation channel.

As shown in Fig. B.8, four passive ENodes and one active ENode have been composed together in this particular experimentation channel. Packets that are conveyed through the return link EChannel cross the four ENodes. Each ENode is responsible for a particular aspect of the on-demand access class behaviour. All together, they emulate the signalling protocol and the return link access behaviour:

- The first ENode emulates the interval of time between two capacity requests that are sent from the satellite terminal toward the NCC. A spying port is placed

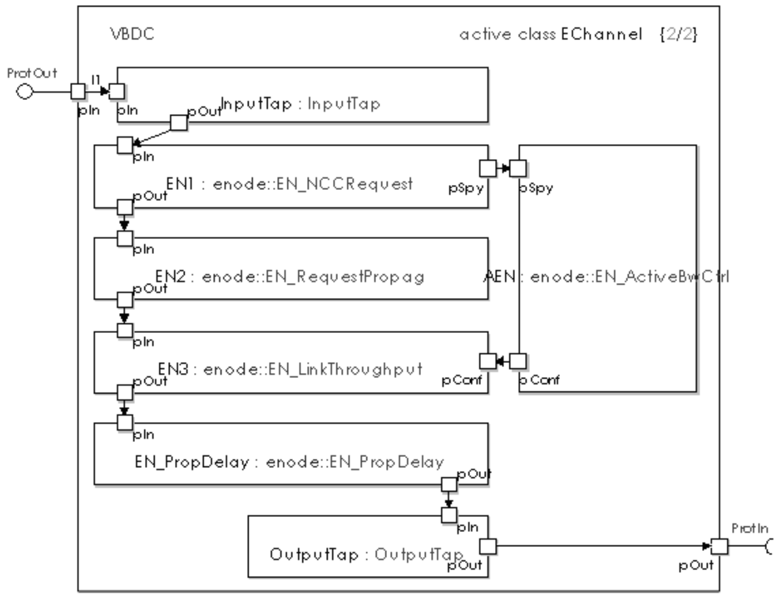
VBDC
active class EChannel {2/2}
ProtOut
l1
pIn
pIn
pOut
InputTap : InputTap
pIn
EN1 : enode::EN_NCCRequest
pOut
pSpy
pSpy
pIn
EN2 : enode::EN_RequestPropag
pOut
AEN : enode::EN_ActiveBwCtrl
pIn
EN3 : enode::EN_LinkThroughput
pOut
pConf
pConf
pIn
EN_PropDelay : enode::EN_PropDelay
pOut
pIn
OutputTap : OutputTap
pOut
ProtIn
pOut

Fig. B.8 On-demand access classes experimentation channel

on this node to measure the volume and rate of incoming traffic and send this information to the emulation control (the active ENode).

- The second ENode emulates the propagation delay from the ST to the NCC of the capacity requests and their processing time. A constant delay is applied with a possible delay variation introduced by request processing.
- The third ENode emulates the emission rate of data on the air link by the ST. This ENode has a communication port and applies bandwidth impairments, depending on information received from the emulation controller (the active ENode). This is based on real-time traffic measurements (volume or rate) by spying traffic on the first ENode. The Active ENode spies on the traffic packets and computes the actual input rate in real-time. The configuration of the rate is calculated when packets reach the first ENode, but needs to be applied when requests come back. As a result, there is a delay introduced between the calculated rate and the time it is applied at the third ENode. The emulated rate can be limited by the SLA limitations as well as emulation of cross-traffic and the satellite system. This ENode also applies the buffer limitations.
- The fourth ENode emulates the delay encountered by the traffic to cross the satellite link. A constant delay is applied corresponding to propagation delay (250 ms).

B.3 Conclusions

Emulation is a widely used approach to experiment real protocols or applications in order to meet user or QoS requirements. In this chapter, we proposed an overview of emulation approaches in the context of networking experiments. We explained why this emulation approach is useful at research, design, and development phases and we compared it to simulation or real network experiments. Emulation is a trade-off between these two approaches, as it provides a way to test real applications or protocols being developed (as opposed to simulation) in a controllable environment (as opposed to real technologies). We defined a general emulation framework that should encompass all the possible emulation platforms. For this, we defined functional requirements of emulation platforms, as well as requirements on packet impairments that such platforms shall be able to provide. We have then described the existing emulation platforms implementing this general framework, classified in two types to realise the traffic shaping: the centralised approach (everything is done on one system) and the distributed approach (the traffic shaping is realised by many computers working together). To control this traffic shaping, a set of emulation models exist with their own advantages and disadvantages: ad-hoc models setting up QoS parameters of the impairment, virtual nodes models, trace-based models, simulation based models and active emulation models. Finally, an emulation example is described, corresponding to a QoS-oriented satellite link integrated in the EuQoS System. This is an example of how to map the technology characteristics toward the emulation model using a centralised emulation approach.

References

1. R. Steinmetz and K. Nahrstedt, *Multimedia Fundamentals: Medai Coding and Content Processing*, 2nd ed. Prentice-Hall PTR, 2002, vol. 1.
2. ——, *Multimedia Systems*, 1st ed. Berlin, Germany: Springer Verlag, 2004.
3. S. Floyd, “Connections with multiple congested gateways in packet-switched networks part 1: One-way traffic,” *Computer Communications Review*, vol. 21, no. 5, pp. 30–47, October 1991.
4. M. Mathis, J. Semke, and J. Mahdavi, “The macroscopic behavior of the tcp congestion avoidance algorithm,” *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, 1997.
5. ITU-T, “Itu t recommendation g.114: One-way transmission time.” 2003.
6. ——, “Itu t recommendation g.1010: End-user multimedia qos categories.” 2001.
7. T. Szigeti and C. Hattingh, *End-to-End QoS Network Design: Quality of Service in LANs, WANs, and VPNs (Networking Technology)*. CISCO Press, 2005.
8. S. Keshav, *An Engineering Approach to Computer Networking*. Addison Wesley, 1997.
9. R. Braden, D. Clark, and S. Shenker, “Integrated services in the internet architecture: an overview,” RFC 1633, June 1994.
10. S. Shenker and J. Wroclawski, “General characterization parameters for integrated service network elements,” RFC 2215, September 1997.
11. J. Wroclawski, “Specification of the Controlled-Load Network Element Service,” RFC 2211 (Proposed Standard), September 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2211.txt
12. S. Shenker, C. Partridge, and R. Guerin, “Specification of Guaranteed Quality of Service,” RFC 2212 (Proposed Standard), September 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2212.txt
13. J. Solomon, “Applicability Statement for IP Mobility Support,” RFC 2005 (Proposed Standard), Oktober 1996. [Online]. Available: http://www.ietf.org/rfc/rfc2005.txt
14. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, “An Architecture for Differentiated Service,” RFC 2475 (Informational), Dezember 1998, updated by RFC 3260. [Online]. Available: http://www.ietf.org/rfc/rfc2475.txt
15. K. Nichols, S. Blake, F. Baker, and D. Black, “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,” RFC 2474 (Proposed Standard), Dezember 1998, updated by RFCs 3168, 3260. [Online]. Available: http://www.ietf.org/rfc/rfc2474.txt
16. K. Nichols and B. Carpenter, “Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification,” RFC 3086 (Informational), April 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3086.txt
17. K. Nichols, V. Jacobson, and K. Poduri, “A per-domain behavior for circuit emulation in ip networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 71–83, 2004.
18. B. Davie, A. Charny, J. Bennet, K. Benson, J. L. Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis, “An Expedited Forwarding PHB (Per-Hop Behavior),” RFC 3246 (Proposed Standard), März 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3246.txt
19. J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, “Assured Forwarding PHB Group,” RFC 2597 (Proposed Standard), June 1999, updated by RFC 3260. [Online]. Available: http://www.ietf.org/rfc/rfc2597.txt
20. J. Babiarz, K. Chan, and F. Baker, “Configuration Guidelines for DiffServ Service Classes,” RFC 4594 (Informational), August 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4594.txt
21. L. Breslau, E. W. Knightly, S. Shenker, I. Stoica, and H. Zhang, “Endpoint admission control: architectural issues and performance,” in *SIGCOMM ’00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. New York, NY, USA: ACM, 2000, pp. 57–69.
22. K. Ramakrishnan and S. Floyd, “A Proposal to add Explicit Congestion Notification (ECN) to IP,” RFC 2481 (Experimental), Januar 1999, obsoleted by RFC 3168. [Online]. Available: http://www.ietf.org/rfc/rfc2481.txt

23. C. Cetinkaya, V. Kanodia, and E. W. Knightly, "Scalable services via egress admission control," *IEEE Transactions on Multimedia*, vol. 3, no. 1, pp. 69–81, 2001. [Online]. Available: citeseer.ist.psu.edu/cetinkaya01scalable.html
24. S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *ACM SIGCOMM*, vol. 26, no. 4. New York: ACM Press, Aug. 1996, pp. 117–130.
25. V. Paxson, "Strategies for Sound Internet Measurement," in *4rth Internet Measurements Conference*, 2004.
26. V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, "Framework for IP Performance Metrics," RFC 2330, May 1998.
27. J. Mahdavi and V. Paxson, "IPPM Metrics for Measuring Connectivity," RFC 2678, Sept. 1999.
28. G. Almes, S. Kalidindi, and M. Zekauskas, "A One-way Delay Metric for IPPM," RFC 2679, Sept. 1999.
29. I.-T. R. Y.1541, "Network Performance Objectives fo IP-based Services," Review Jan 2005.
30. G. Almes, S. Kalidindi, and M. Zekauskas, "A Round-trip Delay Metric for IPPM," RFC 2681, Sept. 1999.
31. ——, "A One-way Packet Loss Metric for IPPM," RFC 2680, Sept. 1999.
32. C. Demichelis and P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)," RFC 3393, Nov. 2002.
33. M. Mathis and M. Allman, "A Framework for Defining Empirical Bulk Transfer Capacity Metrics," RFC 3148, July 2001.
34. I.-T. R. P.800.1, "Mean Opinion Score (MOS) terminology," 2003.
35. A. Takahashi, H. Yoshino, and N. Kitawaki, "Perceptual qos assessment technologies for voip," *IEEE/Comm. Mag*, vol. 42, no. 7, pp. 28–34, 2004.
36. I.-T. R. G.107, "The E-model, a computational model for use in transmission planning," 2003.
37. I.-T. R. P.910, "Subjective video quality assessment methods for multimedia applications," 1999.
38. I.-T. R. P.862, "Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs," 2001.
39. D. L. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC 1305, Mar. 1992.
40. D. Veitch, S. Babu, and A. Pàsztor, "Robust synchronization of software clocks across the internet," in *Internet Measurement Conference (IMC '04)*, 2004.
41. N. Duffeld, "Sampling for passive internet measurement: A review," *Statistical Science*, vol. 19, no. 3, pp. 472–498, 2004.
42. J. Quittek, T. Zseby, B. Claise, and S. Zander, "Requirements for IP Flow Information Export (IPFIX)," RFC 3917, Oct. 2004.
43. M. Crovella and B. Krishnamurthy, *Internet Measurement. Infrastructure, Traffic, and Applications*. John Wiley & Sons, Ltd, 2006.
44. D. Harrington, R. Presuhn, and B. Wijnen, "An architecture for describing simple network management protocol (snmp) management frameworks," RFC 3411, Dec. 2002.
45. "Lobster - Large-scale Monitoring of Broadband Internet Infrastructures," http://www.ist-lobster.org.
46. N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, pp. 280–292, 2001.
47. T. Zseby, S. Zander, and G. Carle, "Evaluation of Building Blocks for Passive One-Way-Delay Measurements," in *Passive and Active Measurements Conference*, 2001.
48. S. B. Moon, "Measurement and Analysis of End-to-end Delay and Loss in the Internet," Ph.D. dissertation, University of Massachusetts, 2000.
49. J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Internet Measurement Conference (IMC)*, 2003.

50. L. Lao, C. Dovrolis, and M. Y. Sanadidi, “The probe gap model can underestimate the available bandwidth of multihop paths,” *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 5, pp. 29–34, 2006.
51. B. Melander, M. Bjorkman, and P. Gunningberg, “A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks,” in *IEEE Globecom Global Internet Symposium*, 2000.
52. Q. Liu and J.-N. Hwang, “End-to-end available bandwidth estimation and time measurement adjustment for multimedia qos,” in *ICME ’03: Proceedings of the 2003 International Conference on Multimedia and Expo - Volume 3 (ICME ’03)*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 373–376.
53. A. Pásztor and D. Veitch, “Active Probing using Packet Quartets,” in *Internet Measurement Workshop*, 2002.
54. R. Jain and S. A. Routhier, “Packet Trains-Measurements and a New Model for Computer Network Traffic,” *IEEE Journal of Selected Areas in Communications*, vol. SAC-4, no. 6, pp. 986–995, 1986.
55. C. Dovrolis, P. Ramanathan, and D. Moore, “Packet-dispersion techniques and a capacity-estimation methodology,” *IEEE/ACM Trans. Netw.*, vol. 12, no. 6, pp. 963–977, 2004.
56. “PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services,” http://www.planet-lab.org/.
57. “Cooperative Association for Internet Data Analysis (CAIDA),” http://www.caida.org/.
58. “Active Measurement Project (AMP),” http://amp.nlanr.net/.
59. F. Georgatos, F. Gruber, D. Karrenberg, M. Santcroos, A. sanj, H. Uijterwaal, and R. Wilhem, “Providing Active Measurements as a Regular Service for ISP’s,” in *Proceedings of Passive and Active Measurement*, 2001.
60. “Evergrow Traffic Observatory Measurement InfrastruCture,” http://www.etomic.org/.
61. “DIMES (Distributed Internet MEasurements & Simulations),” http://www.netdimes.org/.
62. “Ever-growing global scale-free networks, their provisioning, repair and unique functions,” http://www.evergrow.org/.
63. S. Shalunov, B. Teitelbaum, A. Karp, J. W. Boote, and M. J. Zekauskas, “One-Way Active Measurements Protocol,” RFC 4656, Sept. 2006.
64. S. McCanne and V. Jacobson, “The BSD Packet Filter: A New Architecture for User-level Packet Capture,” in *USENIX Winter*, 1993.
65. “perfSONAR - PERFormance Service-Oriented Network monitoring ARchitecture,” http://www.perfsonar.net.
66. I. Miloucheva, P. Gutierrez, D. Hetzer, A. Nassri, and M. Beoni, “Intermon architecture for complex QoS analysis in inter-domain environment based on discovery of topology and traffic impact,” in *Inter-domain Performance and Simulation Workshop, Budapest*, March 2004.
67. “Passive Measurement and Analysis (PMA),” http://pma.nlanr.net/.
68. D. Awduche, J. Malcolm, J. Agogbua, M. O’Dell, and J. McManus, “Requirements for Traffic Engineering Over MPLS,” RFC 2702 (Informational), Sept. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2702.txt
69. E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, and A. Conta, “MPLS Label Stack Encoding,” RFC 3032 (Proposed Standard), Jan. 2001, updated by RFCs 3443, 4182. [Online]. Available: http://www.ietf.org/rfc/rfc3032.txt
70. L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas, “LDP Specification,” RFC 3036 (Proposed Standard), Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3036.txt
71. H. Smit and T. Li, “Intermediate System to Intermediate System (IS-IS) Extensions for Traffic Engineering (TE),” RFC 3784 (Informational), June 2004, updated by RFC 4205. [Online]. Available: http://www.ietf.org/rfc/rfc3784.txt
72. D. Katz, K. Kompella, and D. Yeung, “Traffic Engineering (TE) Extensions to OSPF Version 2,” RFC 3630 (Proposed Standard), Sept. 2003, updated by RFC 4203. [Online]. Available: http://www.ietf.org/rfc/rfc3630.txt

73. R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification,” RFC 2205 (Proposed Standard), Sept. 1997, updated by RFCs 2750, 3936, 4495. [Online]. Available: http://www.ietf.org/rfc/rfc2205.txt
74. D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, “RSVP-TE: Extensions to RSVP for LSP Tunnels,” RFC 3209 (Proposed Standard), Dec. 2001, updated by RFCs 3936, 4420, 4874. [Online]. Available: http://www.ietf.org/rfc/rfc3209.txt
75. B. Jamoussi, L. Andersson, R. Callon, R. Dantu, L. Wu, P. Doolan, T. Worster, N. Feldman, A. Fredette, M. Girish, E. Gray, J. Heinanen, T. Kilty, and A. Malis, “Constraint-Based LSP Setup using LDP,” RFC 3212 (Proposed Standard), Jan. 2002, updated by RFC 3468. [Online]. Available: http://www.ietf.org/rfc/rfc3212.txt
76. D. Grossman, “New Terminology and Clarifications for Diffserv,” RFC 3260 (Informational), Apr. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3260.txt
77. F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen, “Multi-Protocol Label Switching (MPLS) Support of Differentiated Services,” RFC 3270 (Proposed Standard), May 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3270.txt
78. F. L. Faucheur and W. Lai, “Requirements for Support of Differentiated Services-aware MPLS Traffic Engineering,” RFC 3564 (Informational), July 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3564.txt
79. F. Le Faucheur and W. Lai, “Maximum Allocation Bandwidth Constraints Model for Diffserv-aware MPLS Traffic Engineering,” RFC 4125 (Experimental), June 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4125.txt
80. F. Le Faucheur, “Russian Dolls Bandwidth Constraints Model for Diffserv-aware MPLS Traffic Engineering,” RFC 4127 (Experimental), June 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4127.txt
81. W. Lai, “Bandwidth Constraints Models for Differentiated Services (Diffserv)-aware MPLS Traffic Engineering: Performance Evaluation,” RFC 4128 (Informational), June 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4128.txt
82. F. Le Faucheur, “Protocol Extensions for Support of Diffserv-aware MPLS Traffic Engineering,” RFC 4124 (Proposed Standard), June 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4124.txt
83. H. Schulzrinne and R.Hancock, “General Internet Messaging Protocol for Signaling,” IETF, Tech. Rep., 2006.
84. J. Manner, G. Karagiannis, A. McDonald, and S. V. den Bosch, “NSLP for Quality-of-Service Signaling,” IETF, Tech. Rep., 2005.
85. M. Stiemerling, “Loose End Message Routing Method for NATFW NSLP,” IETF, Tech. Rep., 2005.
86. R. Stewart, Q. Xie, K. Morneault, and H. Schwarzbauer, “Stream Control Transmission Protocol,” IETF, Tech. Rep. 2960, 2000.
87. T. Dierks and C. Allen, “The TLS Protocol,” IETF, Tech. Rep. 2246, 1999.
88. D. Katz, “Router Alert Option,” IETF, Tech. Rep. 2113, 1997.
89. M. Stiemerling, H. Tschofenig, C. Aoun, and E. Davies, “Nat/firewall nsis signaling layer protocol (nslp),” IETF,” internet draft, 2006.
90. D.Durham, J.Boyle, R.Cohen, S.Herzog, R.Rajan, and A.Sastry, “The COPS (Common Open Policy Service) Protocol,” IETF, Tech. Rep. 2748, 2000.
91. isox214, “ITU-T recommendation X.214 (11/95) information technology,” Nov. 1995.
92. J. Postel, “Transmission control protocol: DARPA internet program protocol specification,” IETF, Request For Comments 793, 1981.
93. ——, “User datagram protocol (UDP),” IETF, Request For Comments 768, Aug. 1980.
94. H.Schulzrinne, S.Casner, R. Frederic, and V. Jacobson, “An extension to the Selective Acknowledgment (SACK) Options for TCP,” IETF, Task Force 3550, 2003.
95. V. Jacobson and R. Braden, “TCP extensions for long-delay paths,” IETF, Request For Comments 1072, Oct. 1988.
96. V. Jacobson, R. Braden, and D. Borman, “Tcp extensions for high performance,” IETF, Request For Comments 1323, 1992.

97. L. S. Brakno and L. L. Peterson, “TCP Vegas: End to End congestion Avoidance on a Global Internet,” *IEEE Journal on Selected Areas in Communications*, no. 13, pp. 1465–1480, 1995.
98. S. Floyd, T. Henderson, and A. Gurtov, “The newreno modification to tcp’s fast recovery algorithm,” IETF, Request For Comments 3782, Apr. 2004.
99. S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, “An extension to the selective acknowledgement (SACK) option for TCP,” IETF, Request For Comments 2883, July 2000.
100. O. Ait-Hellal and E. Altman, “Analysis of tcp-vegas and tcp-reno,” in *Proc. of the IEEE International Conference on Communications - ICC*, 1997, pp. 495–499.
101. V. Jacobson, L. Peterson, L. Brakmo, and S. Floyd, “Problems with arizona’s vegas,” mailing list, end2end-tf, Tech. Rep., 1994, ftp://ftp.ee.lbl.gov/email/vanj.94mar14.txt.
102. L. Xu, K. Harfoush, and I. Rhee, “Binary increase congestion control for fast long-distance networks,” in *Proc. of IEEE INFOCOM*, Mar. 2004, pp. 2514–2524.
103. S. Floyd, “Highspeed tcp for large congestion windows,” IETF, Request For Comments 3649, Dec. 2003.
104. D. Katabi, M. Handley, and C. Rohrs, “Congestion control for high bandwidth-delay product networks,” in *Proc. of ACM SIGCOMM*, 2002.
105. Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, “One more bit is enough,” in *Proc. of ACM SIGCOMM*, 2005.
106. K. Ramakrishnan, S. Floyd, and D. Black, “The addition of explicit congestion notification (ECN) to ip,” IETF, Request For Comments 3168, Sept. 2001.
107. P. Sinha, T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, “WTCP: A reliable transport protocol for wireless wide-area networks,” *Wireless Networks*, vol. 8, no. 2-3, pp. 301–316, 2002.
108. M. Gerla, M. Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo, “Tcp westwood: congestion window control using bandwidth estimation,” in *Proc. of IEEE GLOBECOM*, 2001, pp. 1698–1702.
109. C. P. Fu and S. C. Liew, “Tcp veno: Tcp enhancement for transmission over wireless access networks,” *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 2, pp. 216–229, Feb. 2003.
110. R. Stewart, Q. Xie, L. Yarroll, J. Wood, K. Poon, K. Fujita, and M. Tuexen, “Sockets API extensions for stream control transmission protocol (SCTP),” IETF, Internet Draft draft-ietf-tsvwg-sctpsocket-06.txt, Mar. 2003.
111. E. Kohler, M. Handley, and S. Floyd, “Datagram congestion control protocol (DCCP),” IETF, Request For Comments 4340, Mar. 2006.
112. S. Floyd and E. Kohler, “Profile for DCCP Congestion Control ID 2: TCP-like Congestion Control,” IETF, Request For Comments 4341, Mar. 2006.
113. S. Floyd, E. Kohler, and J. Padhye, “Profile for DCCP congestion control ID 3: TRFC congestion control,” IETF, Request For Comments 4342, Mar. 2006.
114. M. Handley, S. Floyd, J. Pahdye, and J. Widmer, “Tcp-friendly rate control (TFRC): Protocol specification,” IETF, Request For Comments 3448, Jan. 2003.
115. S. Iren, P. D. Amer, and P. T. Conrad, “The transport layer: tutorial and survey,” *ACM Computer Survey*, vol. 31, no. 4, pp. 360–404, 1999.
116. S. Floyd, “Congestion control principles,” IETF, Request For Comments 2914, Sept. 2000.
117. Y.-Q. Z. D. Wu, T. Hou, “Transporting real-time video over the internet: Challenges and approaches,” in *Proc. of the IEEE*, vol. 88, no. 12, Dec. 2000, pp. 1855–1875.
118. V. Jacobson, “Congestion avoidance and control,” in *Proc. of ACM SIGCOMM*, Stanford, CA, Aug. 1988, pp. 314–329.
119. R. Talluri, “Error-resilience video coding in the ISO MPEG-4 standard,” *IEEE Communications Magazine*, pp. 112–119, June 1998.
120. Y. Wang and Q.-F. Zhu, “Error control and concealment for video communication: A review,” in *Proc. of the IEEE*, vol. 86, no. 5, May 1998, pp. 974–997.
121. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP selective acknowledgment options,” IETF, Request For Comments 2018, Oct. 1996.
122. J. Widmer, “Equation-based congestion control,” Diploma Thesis, University of Mannheim, Germany, Feb. 2000.

123. S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, 1999.
124. P. Amer, C. Chassot, C. Connolly, P. Conrad, and M. Diaz, "Partial order transport service for MM and other application," *IEEE/ACM Transactions on Networking*, vol. 2, no. 5, 1994.
125. L. Rojas-Cardenas, E. Chaput, L. Dairaine, P. Snac, and M. Diaz, "Video transport over partial order connections," *Computer Networks*, vol. 31, no. 7, pp. 709–725, Apr. 1999.
126. J.-P. V. A. Farrel and J. Ash, "A Path Computation Element (PCE)-Based Architecture," The Internet Society, Request For Comments 4655, Aug. 2006.
127. J. Enríquez, M. A. Callejo, and et al., "EuQoS Architecture Deliverable D122," 2007.
128. A. Beben, "EQ-BGP: an efficient inter-domain QoS routing protocol," in *AINA '06: Proceedings of the 20th IEEE International Conference on Advanced Information Networking and Applications*. Los Alamitos, CA, United States: IEEE Computer Society, Apr. 2006, pp. 560–564.
129. X. Masip-Bruin, M. Yannuzzi, R. Serral-Gracia, J. Domingo-Pascual, J. Enriquez-Gabeiras, M. Callejo, M. Diaz, F.Racaru, G. Stea, E. Mingozzi, A. Beben, W. Burakowski, E. Monteiro, and L. Cordeiro, "The EuQoS System: A solution for QoS Routing in Heterogeneous Networks," *IEEE Communications Magazine*, vol. 45, no. 2, Feb. 2007.
130. Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," The Internet Society, Request For Comments 4271, Jan. 2006.
131. L. Baresse and et al., "Integrated EuQoS system Software architecture for application use cases and API for application driving Deliverable D421."
132. P. Krawiec and W. Burakowski, "Resource allocation strategies for new connections in qos multi-domain networks with signaling capabilities," in *Proceedings of Australian Telecommunication Networks and Applications Conference 2007 (ATNAC 2007)*, Christchurch, New Zealand, Dec. 2007, pp. 337–342.
133. A. Beben and et al., "Definition of measurement and monitoring system for Phase 2 Deliverable D222."
134. A. Bak, W. Burakowski, F. Ricciato, S. Salsano, and H. Tarasiuk, "A framework for providing differentiated QoS guarantees in IP-based network," *Computer Communications*, vol. 26, no. 4, pp. 327–337, 2003.
135. C. Brandauer, W. Burakowski, M. Dabrowski, B. Koch, and H. Tarasiu, "AC algorithms in AQUILA QoS IP network," *European Transactions on Telecommunications*, vol. 16, no. 3, pp. 225–232, 2005.
136. M. Dabrowski, G. Eichler, M. Fudala, D. Katzengruber, T. Kilkanen, N. Miettinen, H. Tarasiuk, and M. Titze, "Evaluation of the AQUILA Architecture: Trial Results for Signalling Performance, Network Services and User Acceptance," in *Art-QoS*, 2003, pp. 218–233.
137. K.Chan, J.Babiarz, and F.Baker, "Aggregation of DiffServ Service Classes," Transport Area Working Group," Internet-Draft, Nov. 2007.
138. "Nexuiz project," www.alientrap.org/nexuiz/.
139. J. M. Batalla and R. Janowski, "Provisioning dedicated class of service for reliable transfer of signaling traffic," in *International Teletraffic Congress*, ser. Lecture Notes in Computer Science, L. Mason, T. Drwiega, and J. Yan, Eds., vol. 4516. Springer, 2007, pp. 853–864.
140. "Medigraf," http://www.medigraf.pt/.
141. J. W. Roberts, U. Mocci, and J. T. Virtamo, Eds., *Broadband Network Teletraffic - Performance Evaluation and Design of Broadband Multiservice Networks: Final Report of Action COST 242*, ser. Lecture Notes in Computer Science. Springer, 1996, vol. 1155.
142. L. Kleinrock, *Queueing Systems, Volume 2, Computer Applications*. John Wiley & Sons Inc, 1976.
143. H. Tarasiuk, R. Janowski, and W. Burakowski, "Admissible traffic load of real time class of service for inter-domain peers," in *Proceedings of Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS/ICNS 2005)*. Papeete, Tahiti, French Polynesia: IEEE Computer Society, Oct. 2005.
144. ——, "Application of Admission Control and Traffic Shaping for providing TCP Throughput Guarantees," in *Proceedings of International Workshop To-QoS'2006 in conjunction with IFIP 2006 Networking Conference (ed. W.Burakowski)*, Coimbra, Portugal, May 2006, pp. 163–172.

145. “IEEE standards for local and metropolitan area networks. Virtual bridged local area networks,” *IEEE Std 802.1Q, 2003 Edition (Incorporates IEEE Std 802.1Q-1998, IEEE Std 802.1u-2001, IEEE Std 802.1v-2001, and IEEE Std 802.1s-2002)*, 2003.
146. “IEEE Standard for Local and metropolitan area networks Media Access Control (MAC) Bridges,” *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, 2004.
147. M. Carmo, J. S. Silva, E. Monteiro, P. Simões, and F. Boavida, “Ethernet QoS Modeling in Emerging Scenarios,” in *Proceedings of 3rd International Workshop on Internet Performance, Simulation, Monitoring and Measurement (IPS-MoMe 2005)*. Warsaw, Poland: IST MoMe Cluster, Mar. 2005, pp. 90–96.
148. M. Carmo, B. Carvalho, J. S. Silva, E. Monteiro, P. Simões, M. Curado, and F. Boavida, “NSIS-based Quality of Service and Resource Allocation in Ethernet Networks,” in *Proceedings of 4th International Conference on Wired/Wireless Internet Communications 2006*, 2006.
149. M. Carmo, J. S. Silva, and E. Monteiro, “EuQoS approach for Resource Allocation in Ethernet Networks,” *International Journal of Network Management*, vol. 17, no. 5, pp. 373–388, sep / oct 2007.
150. “Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements,” *IEEE Std 802.11e-2005*, 2005.
151. E. Lochin and L. D. A. Jourjon, “gtfrc, a tcp friendly qos-aware rate control for diffserv assured service,” *Springer Telecommunication Systems*, vol. 10.1007/s11235-006-9004-2, ISSN: 1018-4864 (Print) 1572-9451 (Online), September 2006 2006.
152. E. Exposito, P. Sénac, and M. Diaz, “Compositional architecture pattern for qos-oriented communication mechanisms,” in *MMM*, Y.-P. P. Chen, Ed. IEEE Computer Society, 2005, pp. 413–420.
153. B. Quinn and K. Almeroth, “Ip multicast applications: Challenges and solutions,” The Internet Society, Request For Comments 3170, Sept. 2001.
154. Z. Albanna, K. Almeroth, D. Meyer, and M. Schipper, “Iana guidelines for ipv4 multicast address assignments,” The Internet Society, Request For Comments 3171, Aug. 2001.
155. R. Hinden and S. Deering, “Internet protocol version 6 (ipv6) addressing architecture,” The Internet Society, Request For Comments 3513, Apr. 2003.
156. B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, “Internet group management protocol, version 3,” The Internet Society, Request For Comments 3376, Oct. 2002.
157. A. Adams, J. Nicholas, and W. Siadak, “Protocol independent multicast - dense mode (pim-dm): Protocol specification (revised),” The Internet Society, Request For Comments 3973, Jan. 2005.
158. B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, “Protocol independent multicast - sparse mode (pim-sm): Protocol specification (revised),” The Internet Society, Request For Comments 4601, Aug. 2006.
159. D. Waitzman, C. Partridge, and S. Deering, “Distance vector multicast routing protocol,” Network Working Group, Request For Comments 1075, Nov. 1988.
160. J. Moy, “Multicast extensions to ospf,” Network Working Group, Request For Comments 1584, Mar. 1994.
161. K. Savetz, N. Randall, and Y. Lepage, *MBONE: Multicasting Tomorrow’s Internet*. John Wiley & Sons Inc, 1996.
162. R. Zhang and Y. C. Hu, “Borg: A hybrid protocol for scalable application-level multicast in peer-to-peer networks,” in *NOSSDAV ’03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, ACM. New York, NY, USA: ACM Press, June 2003, pp. 172–179.
163. A. Sobeih, W. Yurcik, and J. C. Hou, “Vring: A case for building application-layer multicast rings (rather than trees),” in *MASCOTS ’04: Proceedings of the The IEEE Computer Society?s 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS’04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 437–446.