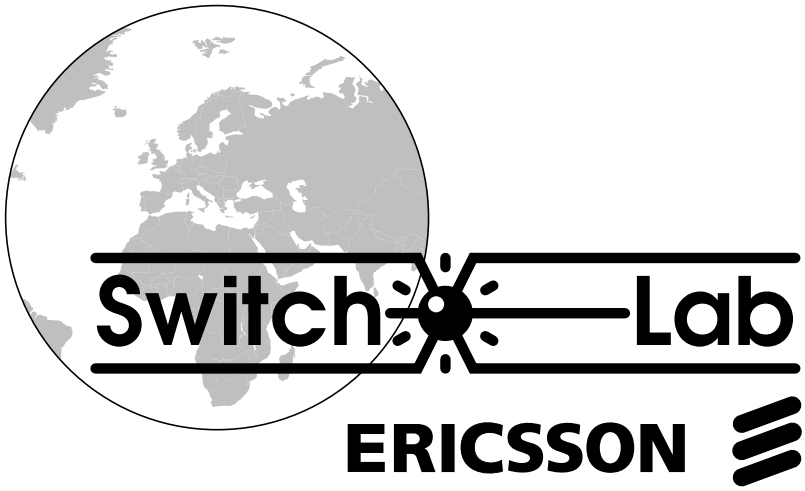


# TESTBED FOR MEASUREMENT BASED TRAFFIC CONTROL

**Mikael Torneus**

Royal Institute of Technology, Sweden
Master's Thesis in Computer Science and Technology

June 2000

***Supervisior:***

Harald E. Brandt
SwitchLab
Ericsson Radio Systems AB

***Examiner:***

Gunnar Karlsson
Department of Teleinformatics
Royal Institute of Technology, Sweden

## ABSTRACT

The work of this master thesis is divided into two parts:

- to build a multi-service testbed based on commercial products, and
- to provide means to predict the available capacity in routers.

A testbed has been built to support multi-services with additional test equipment connected. The additional equipment is able to collect and issue measurements from network elements and other external equipment connected to the testbed. Commercial routers have the ability to support traffic in several service classes with limited support. There are only a few options available to differentiate traffic into service classes and to treat them according to the expected service.

Attempts were made by four methods to predict the available capacity of a router. They all showed different kinds of limitations, and thus, none of the methods could be used for our purpose in the way we intended. The methods were only applicable to one service class, and thus, could not help us to measure the capacity for more classes.

During the attempts to predict the capacity, we discovered a severe limitation in processor capacity of the router when priority queueing was enabled. This limitation reduced the bandwidth capacity of the router by 84% and showed a very unstable behavior of the limitation. A small increase in offered load made the router rejecting almost all traffic.

# *CONTENTS*

**1 Introduction . . . . . 1**
1.1 Background . . . . . 1
1.2 Requirements on Real-time Networks . . . . . 1
1.3 Network Management for IP networks . . . . . 2
1.4 Differentiated Services . . . . . 3
1.5 Objective . . . . . 4
1.6 Disposition . . . . . 5
**2 An Overview on Network Measurement and Methodology . . . . . 7**
2.1 Network Measurement Methodologies . . . . . 7
2.2 Network Measurements . . . . . 8
2.2.1 *Standardized Performance Metrics* . . . . . 8
2.2.2 *Sampling Methodologies* . . . . . 9
2.2.3 *An Internet Measurement Infrastructure* . . . . . 9
2.2.4 *Internet Traffic Measurements* . . . . . 10
**3 Building a Testbed . . . . . 13**
3.1 Testbed Router . . . . . 13
3.1.1 *Scheduling Schemes* . . . . . 13
3.1.2 *Packet Classification by Access Lists* . . . . . 16
3.1.3 *Packet Marking* . . . . . 16
3.2 Tools Used in the Testbed . . . . . 16
3.2.1 *A Traffic Generator* . . . . . 16
3.2.2 *A Network Management Application* . . . . . 17
3.2.3 *A Delay Measurement Tool* . . . . . 18
3.3 A Working Testbed . . . . . 19
**4 Problems in Predicting Available Capacity . . . . . 23**
4.1 Test Traffic . . . . . 23
4.2 Queue Monitoring . . . . . 24
4.3 Processor Load . . . . . 26
4.4 IP and Interface Information . . . . . 32
4.5 Alarm Setting . . . . . 32
**5 Conclusions . . . . . 33**
5.1 Suggestions for Further Studies . . . . . 34
**6 References . . . . . 35**
**Appendix . . . . . 37**

Abbreviations . . . . . 37
A Scotty Script . . . . . 38
Router Configuration . . . . . 46

# 1 INTRODUCTION

## 1.1 *Background*

Today's Internet only provides traditional best-effort service, i.e. transmit as much as possible with all traffic treated equally. Best effort service can mainly be used to carry traffic consisting of file transfers i.e. FTP, HTTP, e-mail etc. But lately, an increased demand for real-time applications on the Internet has arisen which puts other requirements on the network and its services.

Several schemes have been proposed to change the Internet into a more commercial infrastructure that supports real-time services e.g telephony and video. One of the most interesting schemes is the Differentiated Services scheme. Differentiated Services give e.g. time sensitive traffic priority over traffic which does not have these requirements.

By changing the Internet from only a single-service to a multi-service network, the network capacity for the different providers will be a commodity. This commodity will be sold and bought among the providers to fulfill users' demands. To be able to sell capacity in a network, it must be predicted in some way. A convenient and economical way would be to use the already existing infrastructure for the network management system.

This master thesis will investigate if it is possible to predict available capacity, by using an existing infrastructure, in a multi-service IP network.

## 1.2 *Requirements on Real-time Networks*

Requirements for real-time networks can basically be divided into four components:

- Delay
- Jitter
- Packet loss rate
- Bandwidth efficiency

A one-way delay of 150 milliseconds is often stated as a maximum requirement for having a fluent conversation [25]. This maximum delay corresponds to the total end-to-end delay from mouth to ear which includes delays in network and end systems. A network delay consists of a transmission delay, a minimum forwarding delay as well as a variable forwarding and buffering delay. The transmission and the minimum forwarding delays are regarded as a fixed network delay with no delay variation. The variable forwarding and buffering delay is caused by route changes and queueing. In end systems, codecs are needed which add a constant delay to the total end-to-end delay.

Jitter, i.e. delay variation, is caused by the packet network. This can be solved by delay equalization at the receiver end system. A delay equalization introduces additional delay to the total delay but the end user will experience an approximately constant delay.

Packet loss is often related to audio quality. Audio processing for telephony can be designed to be relatively tolerant to packet loss. A sufficient audio quality can be achieved at a packet loss of 1%.

Bandwidth efficiency is only important in areas where bandwidth is a scarce resource, e.g. a wireless access for mobile users. Bandwidth efficiency for short packets (88 bytes) is poor due to packet headers (IP/UDP/RTP). By using header compression, the packet headers can be reduced from 40 bytes to 3-5 bytes which is a big improvement in bandwidth efficiency.

A maximum round-trip delay of 300 milliseconds is needed for a fluent conversation. In a GSM network, a standardized maximum round-trip delay is specified to 188.5 milliseconds [25]. This translates to a remaining delay budget of 111.5 milliseconds for connected networks. If one of the connected networks is an IP core network, an IP gateway for speech coding is needed. This will subtract additional 40 milliseconds from the remaining delay budget which ends up in approximately 70 milliseconds. If parts of the GSM system migrate to IP-based transmission for better bandwidth utilization, additional delays will reduce the delay budget.

This is worst case figures but the IP core network with its end systems should be designed to not consume more than what the delay budget allows.

## 1.3 *Network Management for IP networks*

Network management for IP networks consist of one or more network managers and a number of network agents distributed throughout the network. The agents are implemented in network elements, i.e. routers and printers, while a manager is normally implemented in a workstation with a graphical user interface. Managers communicate with agents to retrieve or set information. Agents may notify managers if something happens in the network element.

Network management for IP networks [13] can be divided into three pieces:

- Management Information Base (MIB)
- Structure of Management Information (SMI)
- Simple Network Management Protocol (SNMP)

A MIB specifies what variables a network element maintains. This information can be retrieved and set by a SNMP manager. Two types of MIBs exist, standard and enterprise MIBs. Standard MIBs are defined by RFCs and should be implemented by all network elements. Enterprise MIBs maintain variables which are specific for a certain vendor or even a particular model, e.g. how to access buffers or queues in a network element. These MIB variables can not be used by equipment from other vendors. New MIBs, both standard and enterprise, are continuously developed to support new functionality. All MIB variables are defined using Abstract Syntax Notation One (ASN.1).

SMI describes a common structure and identification scheme to reference variables in a MIB. It specifies, among other things, how data types should be interpreted and MIB variables be identified. SMI specifies several data types, e.g. an OCTET STRING is a string of

zero or more 8-bit bytes or an IpAddress is an OCTET STRING of length 4, with one byte for each byte of the IP address. It also specifies how a variable is identified. A variable is given a unique name, an OBJECT IDENTIFIER, which is a sequence of integers, e.g. 1.3.6.1.2.1.7 is variable iso.org.dod.internet.mgmt.mib.udp.udpInDatagrams. OBJECT IDENTIFIER is based on a tree structure, similar to DNS, which is specified by authoritative organizations.

The SNMP protocol defines the messages and formats sent between a manager and an agent or between two managers. SNMP is normally used with UDP which means that a manager should have a retransmission functionality implemented in case messages are lost. The messages are of type *get information*, *send information* and *traps*. Get *information* messages are sent from a manager to an agent and the agent replies with *send information*. *Traps* are only used by an agent to notify a manager when an extraordinary event has occurred, e.g. a link is down.

## 1.4 *Differentiated Services*

The Internet Engineering Task Force (IETF) has been working for years to find a mechanism to achieve Quality of Service (QoS) in IP networks. One solution was Integrated Services (IntServ). IntServ provide QoS on a per-flow basis, which means that every router on the path from sender to receiver has to keep state information per flow. This solution obviously does not scale well across a large internetwork like the Internet. To solve the scalability problem that IntServ introduced, another solution was presented, namely Differentiated Services (DiffServ).

DiffServ operates on groups of flows that have similar QoS requirements and do not need to keep information about every flow through the network elements and thus should scale well.

A DiffServ domain (DS domain) is a contiguous set of nodes that support DiffServ as defined in [6]. In a DS domain, most of the complexity is moved towards the border of the domain while the inner part should be kept as simple as possible. By keeping the network elements inside a DS domain simple, it should be possible to make the packet forwarding engine efficient and fast.

The inner part of DS domain consists of DS nodes which must be able to handle:

- packet classification, and
- different forwarding mechanisms.

A packet is classified according to a DiffServ Codepoint (DSCP) which is located in the DS field. The DS field was former the Type of Service field (TOS) found in the IP header. DSCP indicates what type of per-hop behavior (PHB) should be applied to a packet when it traverses a DS domain.

An interior DS node must support several forwarding mechanisms which can be used by different PHBs. A PHB is a packet forwarding treatment and is defined in terms of behavior characteristics relevant to a service, e.g. by delay, loss or bandwidth. A PHB is not a specific

packet forwarding mechanism i.e. several forwarding mechanisms can be suitable for the same PHB or groups of PHBs [6]. Priority Queueing or Weighted Fair Queueing could be used for the same PHB.

At the border of a DS domain, a careful traffic conditioning must be supported. A border router should have the following functionality:

- packet marking,
- packet classification,
- traffic shaping, and
- traffic policing.

When a packet enters a DS domain, it will be marked or remarked by a border router. A packet is marked the first time it enters a DS domain and remarked if the packet comes from another DS domain, i.e. it has already been marked somewhere else. A packet is (re)marked in the DS field [7] with a DSCP in the IP header.

A packet stream is shaped, i.e. a packet within a stream is delayed, to make it conform to a predefined traffic profile. This makes the stream less bursty. A traffic profile can be specified by e.g. bandwidth and burst size [6].

When packets are out-of-profile, i.e. they do not follow the predefined traffic profile, they can either be delayed (shaped) or discarded (policed). This depends on the traffic profile. For example, if a packet is late and it belongs to a time bounded profile, it will be discarded. On the other hand, if the packet is early, it will be delayed until it conforms to the profile.

When a packet enters a DS domain, it will be (re)marked in the IP header with an appropriate DSCP. The DSCP indicates the PHB for the packet. The packet is conformed to a defined traffic profile by means of packet shaping and policing before it is transmitted further into the DS domain. Inside the DS domain, packets are only classified and treated according to the PHB at the ingress, until they reach the next border of the domain.

## 1.5 *Objective*

The objective of this thesis can be divided into two parts:

- to build a working testbed based on commercial products, and
- to provide means to predict the capacity of routers.

The purpose of the testbed is to build a network with commercial products. It should resemble an existing network as much as possible. The testbed should have the interior part of a Differentiated Services network implemented, and it should also be able to collect and issue measurements from network elements and external equipment connected to the testbed. The testbed is also going to support a service for real-time traffic.

The second purpose of this thesis work is to provide methods to measure the available capacity in each service class in routers. The methods should be easy to implement and with a low impact on existing and operative networks.

## 1.6 *Disposition*

The report consists of five chapters including this introduction.

Chapter 2 gives an overview on network measurement and methodology. The chapter also provide measured traffic data from an existing and operative network.

Chapter 3 describes the different components needed for the testbed and how they were configured to achieve a working testbed.

Chapter 4 covers the different methods used to predict the available capacity in a router.

Chapter 5 addresses the problems discovered during this work and also proposals how to solve them.

The thesis concludes with a chapter of references and an appendix with a list of abbreviations, a Scotty script and a description of a router configuration.

# 2 AN OVERVIEW ON NETWORK MEASUREMENT AND METHODOLOGY

## 2.1 *Network Measurement Methodologies*

Methods for measuring network traffic can basically be divided into two major types [19], [20]:

- Active traffic measurement
- Passive traffic measurement

Active traffic measurement, which is also called intrusive measurement, injects probe traffic into a network. The measurement system generates probe packets periodically, sends them into the network and receives probe packets from the network. The probe packets are affected by the status of the network while they are traversing it, i.e. the probe packets should give a reasonable picture how “real” traffic is affected. There are two drawbacks with this approach, one is the extra load on the network, and the second is when the network is affected by the test traffic which cause the measurements to give misleading results.

Passive traffic measurement, which is also called non-intrusive measurement, monitors all traffic in the network and captures information from each packet based on several parameters defined in the measurement system. One vital drawback is the question about privacy and security. Another possible drawback, due to increased network capacity, is the amount of collected statistical data gathered with this method. The amount of data can very easily be unmanageable.

An example of an active traffic measurement technique is described in [21]. The technique is called *packet pair* and was used to estimate the bottleneck bandwidth, i.e. the upper limit on how quickly the network can deliver the senders data to the receiver. The bottleneck comes from the slowest element in the network path.

The *packet pair* technique is when two packets are sent shortly after each other. The time between the packets must be less than the time to transmit the packet through the bottleneck. When the second packet arrives at the bottleneck, it has to queue after the first packet. This will cause the packets to be spread out in time. By knowing the inter arrival time between the two packets and the packet size, the bottleneck bandwidth can be estimated. Several problems can be addressed to the *packet pair* technique:

- Out-of-order delivery
  - Packet pairs delivered out of order destroy the *packet pair* technique.
- Limitations due to clock resolution
  - A receivers clock resolution can introduce errors when estimating the bottleneck bandwidth.
- Changes in bottleneck bandwidth
  - During the connection, the bottleneck bandwidth can change due to route changes or link technology, e.g. activation of additional ISDN channels.
- Multi-channel bottleneck links

– When multiple channels are used, the *packet pair* may use two different parallel channels which gives an incorrect estimate of the bottleneck.

To overcome the shortcomings of *packet pair*, a more robust procedure was developed, *packet bunch modes* (PBM). PBM uses a range of packet bunch sizes and allows also multiple bottlenecks values.

An example of a passive traffic measurement technique is described in [19]. The measurement system consisted of a probe, for capturing packets, an analyzer, for extracting necessary information, and a storage unit for saving all measurement data. The stored measurement data was collected and analyzed to retrieve the characteristics of Internet traffic.

A sound measurement methodology [20] takes into account the measurement context. An unsound methodology does not which could cause erroneous results. A measurement context is explained in Section 2.2.1.

## 2.2 *Network Measurements*

### *2.2.1 Standardized Performance Metrics*

The lack of standardization of Internet measurement has made it very difficult to diagnose network problems, compare measurement data and if promised performance is received or not. By introducing standardized Internet measurements [20], the following can benefit from it:

- Improve trouble-shooting and capacity planning.
- Incentives for network service providers to optimize their network e.g. competitors can be compared.
- Help Internet research community to better understand network traffic and how Internet evolves.

Internet performance metrics can be well-defined or ill-defined. If a metric for a routers forwarding rate is defined as number of packets per second, the size of the packets may be relevant. If the router copies each packet to a temporary buffer and the copying dominates the processing time, the size of the packets will be relevant. If the copying is done in parallel with the forwarding lookup and the lookup takes longer time than copying the maximum sized packet, then the size of a packet is irrelevant. A well-defined metric contains all relevant factors while an ill-defined does not.

A measurements context is an important term and is defined as those elements of the complete system used to make the measurement that are in addition to the component being measured, i.e. it is crucial to understand the surrounding environment of the measurement.

Metrics can be of two types:

- Analytically-specified metrics

  - These metrics are defined in an abstract mathematical way, such as “buffer size of a router”, “propagation time for a link” or “instantaneous route of a network path”.
- Empirically-specified metrics
  - These metrics are defined directly in terms of a measurement methodology. A empirical metric can be for example “the output of the traceroute program run on host A with the argument B”.

There can also be compositions of metrics, e.g. “transmission time of a link” is a composition of “propagation time of a link” and “bandwidth of a network link”. It is generally easier to make composition of analytical metrics compared to empirical due to the increased complexity in empirical metrics.

### 2.2.2 Sampling Methodologies

Due to an increase in network capacity, a larger and larger collection of statistics data is gathered. This amount of data will quickly become unmanageable. One way to reduce the amount of data is to take samples. This section presents various methods of sampling related to wide area network traffic characterization [22].

Three main classes of sampling schemes are presented:

- Systematic sampling
  - Every k:th packet is selected from the data set.
- Stratified random sampling
  - A packet is selected randomly from each bucket. A bucket, in this case, is a collection of measurements and has a constant size.
- Simple random sampling
  - Selects n packets randomly.

Different methods on how to trigger a sample exists, two methods are presented:

- A timer-driven method
  - A timer is used to select a packet. When the timer expires, the next packet arriving is selected.
- An event-driven method
  - A packet count is used to trigger the selection.

The results from [22] show a larger deviation from the original data set and a higher variance when the number of samples was decreased, and in general, the time-driven sampling is worse. It tends to miss bursty periods with many packets of small interarrival times.

### 2.2.3 An Internet Measurement Infrastructure

Studies have shown that the characteristics of Internet traffic varies so much between different networks that there exists no typical Internet traffic [23]. Based on these studies, a

full range measurement infrastructure has been developed to fully characterize the Internet traffic.

The idea is that the measurement platforms cooperate with each other and exchange test traffic to measure the Internet properties along the paths. The measurement platforms should be deployed by the network operators. Incentives for this are:

- Detect and debug performance problems in their own network.
- Check if the operator gets the service stated in the service level agreement with that operator.
- Make measurement platforms available for measuring competitor's networks.

A prototype was developed and the experience gained from this project is concluded in a number of design goals and constraints that operators believe a measurement infrastructure must meet. Some of them are:

- The measurement platforms will be in administratively diverse environments.
- It should support a wide range of measurements.
- Perform active instead of passive measurements, otherwise this can cause privacy and security problems.
- Scalable for many measurement platforms and must not load the network.
- Have a solid security and authentication when accessing the measurement platforms.

Based on some of the requirements and constrains above, a National Internet Measurement Infrastructure (NIMI) has been developed. At the moment, there exists six NIMI platforms at five sites, four in the US and one in Europe. The number of platforms are expected to increase.

### 2.2.4 *Internet Traffic Measurements*

Internet traffic measurements have been performed on several networks. In [19], traffic measurements was performed on the backbone of the Inter-Ministry Network (IMNET), which is part of the Japanese Internet. The traffic on IMNET is from/to the Japanese governments laboratories and research organizations.

A passive traffic measurement method was used with a one hour measurement period. During the one hour measurement, 98.2% of the IP traffic was TCP traffic.

Four TCP packet lengths were identified which were more frequently used:

- 40 bytes packet length
  - This length was caused by TCP ACK packets. TCP ACK packets are control packets and shows that TCP data transfer is usually one-way, from server to client.
- 552 bytes packet length
  - This length was caused by the Maximize Segment Size (MSS). MSS specifies the maximum sized segment the sender wants to receive.
- 576 bytes packet length

  - This length was also caused by MSS.
- 1500 bytes packet length
  - This is the Maximum Transfer Unit (MTU) for the segment.

The UDP packet lengths were approximately 75-81 and 740 bytes. These packet lengths came from Domain Name System (DNS) and Simple Network Management Protocol (SNMP) traffic.

In each TCP session, the following most frequently used data were identified:

- 8 or 16 packets were used.
- Approximately 700 bytes were sent.
- A connection time of approximately 556 milliseconds.

The average loss rate for TCP packets was approximately 4.3%. A packet loss of 5.2% was measured in [21]. A packet loss figure for the whole Internet could be close to these figures.

# 3 BUILDING A TESTBED

## 3.1 *Testbed Router*

The router used in this testbed is a Cisco 2611 modular access router [1]. The Cisco 2600 series is a relatively new office router. This kind of router is typically used in small corporate networks with a limited transmission capacity. It supports new business applications, such as Internet telephony and desktop video. Cisco 2611 has three 10 Mbit/s Ethernet ports and uses the Cisco IOS Release 12.0(5)T software. This section describes Cisco's available scheduling schemes, how packets are classified and how they are marked.

### *3.1.1 Scheduling Schemes*

A scheduling scheme, see Figure 3-1, controls how packets are treated on the egress side of a router, e.g. after the next destination has been determined. A scheduling scheme is often implemented as so called hold queues before the actual transmit queue. The transmit queue always works as a FIFO queue, irrespective of how the hold queues are configured. Prior to the hold queues, a packet classification must be performed to select the appropriate hold queue. The classification is of different granularity and differs between the scheduling schemes.

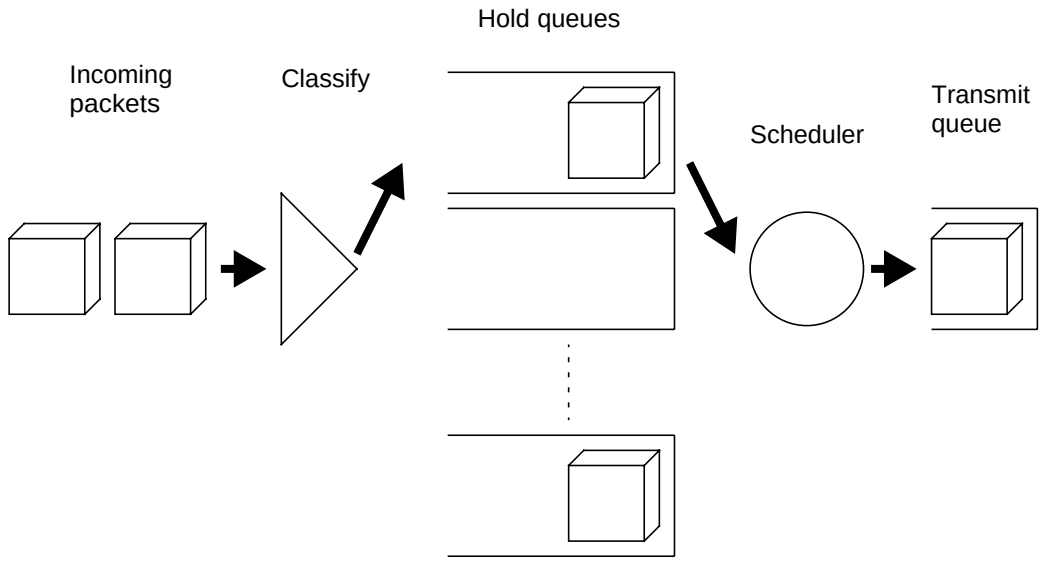


***Figure 3-1 An overview on a scheduling scheme****. Incoming packets destined for an outgoing interface are classified for an appropriated hold queue selection. The packet stays in the hold queue until it will be selected by the scheduler for transmission. The scheduler puts the packet in the transmit queue for further processing at the outgoing interface.*

Cisco 2611 has several scheduling schemes implemented. Each scheme shows rather different characteristics which naturally affects the field of application. Four scheduling schemes are available:

- First-In-First-Out Queueing (FIFO)

- Priority Queueing (PQ)
- Custom Queueing (CQ)
- Weighted Fair Queueing (WFQ)

FIFO is a basic store and forwarding scheme. The algorithm stores each packet in a common queue and forwards them in order of arrival. If the queue fills up, all new packets will be discarded until the queue starts transmitting packets again. In this scheme, all packets are treated equally, there is no concept of priority, and thus, it does not support classes of traffic. FIFO is the scheme mostly used in Internet today.

PQ is a strict priority scheme. It is implemented with four different hold queues: High, Medium, Normal and Low in descending order. Each hold queue is adjustable by specifying the maximum number of packets in the queue. When packets arrive, they are classified to be assigned to the correct hold queue. Packets can be classified on the basis of incoming interface, protocol, fragments, access lists or packet size. Fragments are created when the size of an IP packet is bigger than the MTU of the media. The packet is disassembled into smaller packets (fragments) which could fit the MTU of the media. Only the first fragment contains a UDP/TCP header while the rest of the fragments can not be classified by protocol. A work around is to classify by fragment. An access list is a list of rules which can be applied on every incoming packet. If a rule catches a packet, a decision is made to either permit or deny further processing, i.e. access lists works as a fire wall, see also Section 3.1.2. If it is not possible to classify a packet, the packet is put in the Normal queue. The scheduler always starts with the highest priority queue. When the queue is empty, it continues with the next queue in descending order. If a queue is full, all excess traffic to the queue will be discarded.

CQ is a weighted round-robin scheduling scheme which consists of 17 queues but only 16 queues can be configured. Queue number 0 is a system queue for system information, such as *keep alive* and signaling packets. The system queue is always scheduled and emptied before any other queue. The rest of the 16 queues are served in a round-robin fashion. Each of these queues are served according to a predefined number of bytes before the scheduler continues with the next queue. Each queue transmits at least the number of bytes specified. If the number of bytes of the current packet exceeds the remaining byte count, it will still be sent. Due to the round-robin scheduling, CQ can provide a guaranteed bandwidth in the sense of link utilization. This bandwidth guarantee is not valid in an overloaded link situation and thus should not be compared with schemes for reservation of resources, e.g. RSVP. The classification of packets and the specification of maximum queue size are done in a similar way as for PQ.

WFQ [24] is a flow-based scheme which divides the traffic into two categories: high bandwidth and low bandwidth traffic. Low bandwidth traffic, such as control messages, has priority over high bandwidth traffic which shares the resources according to assigned weights. The weights determine the transmit order for queued high bandwidth packets. Packets are classified into a flow according to: source/destination address, protocol and port/socket number of a session. A flow is assigned its own hold queue. Flows are assigned weights which are affected by the Precedence field in the IP header: a high Precedence value will

give more scheduling time than flows with a lower Precedence value. The Precedence field uses three bits from the Type of Service (ToS) field in the IP header, see Figure 3-2. Eight precedence values are available: value 7 has the highest precedence and value 0 has the lowest precedence. The Precedence field is seldom used in the Internet today.

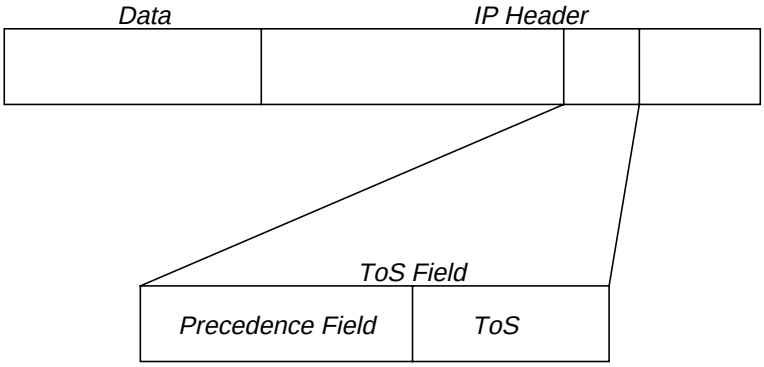


***Figure 3-2 IP version 4 Type of Service field****. Three bits from the ToS field are used as precedence bits. This gives eight available precedence values, Value 7 has the highest and value 0 the lowest precedence value.*

The priority of a flow are calculated from the assigned weight of the flow which actually refers to the precedence bits of the packets of the flow. A weight is assigned by the precedence value plus one. The weight works as a numerator for the link utilization ratio. For example, if it exists one flow for each precedence level, each flow will get precedence plus one utilization ratio of the link:

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36

Thus, precedence 0 traffic will get 1/36 of the bandwidth, precedence 1 traffic will get 2/36 of the bandwidth and precedence 7 traffic will get 8/36 of the bandwidth. However, if it exists 18 precedence 1 flows and one of each of the rest, the utilization ratio will be:

1 + 2 * (18) + 3 + 4 + 5 + 6 + 7 + 8 = 70

Precedence 0 traffic will get 1/70 of the bandwidth and each of the precedence 1 flows will get 2/70 of the bandwidth.

If another degree of priority among the precedence levels is required, this is however not possible. The behavior of WFQ can not be changed in the router software.

The hold queues are adjustable in size by specifying the maximum number of packets in each queue. If a hold queue is full, any additional traffic to that queue will be discarded. This limitation is not valid for low bandwidth traffic where it will continue to enqueue packets in the hold queue.

### 3.1.2 Packet Classification by Access Lists

The above described packet classification for each scheduling scheme may not always be sufficient. The IOS software has additional functionality, e.g. access lists, for a custom-made classification. An access list can identify IP packets on: source/destination address, physical port, protocol, precedence, type of service level or port number. A very fine grained classification and scheduling can be achieved, if access lists and the classification from the scheduling schemes are combined in a appropriate way.

### 3.1.3 Packet Marking

Marking of packets must be possible in a testbed if different traffic characteristics are to be examined. Far from every host has the ability to mark, in an easy way, the Precedence bits in the IP header according to a certain service level. This can easily be done by using Committed Access Rate (CAR) functionality in the router. CAR can, besides marking certain traffic, also rate limit traffic to a specified rate.

CAR examines all incoming traffic, if a packet matches some predefined criteria, an action is performed. A certain type of traffic can be selected by matching on: incoming interface, IP Precedence, MAC address or IP access lists. After a packet is selected by using these criteria, the following actions can be performed: drop a packet, set an IP Precedence bit or just transmit a packet. Several actions can be combined to achieve a suitable functionality. Each criteria is matched sequentially, so the relative order is of importance.

## 3.2 Tools Used in the Testbed

This section describes why and which tools were needed in the testbed. We start with the traffic generator, continue with a network management application, and finally, describe a one-way delay measurement tool.

### 3.2.1 A Traffic Generator

The testbed will simulate a real network with a similar mixture of traffic and load. A traffic generator provides the only reasonable traffic source, since a connection to a real network is neither appropriate nor controllable.

The traffic generator used in this testbed was a SMB-2000 from Netcom Systems [3], called SmartBits. It is a powerful stream-based traffic generator, that supports 10/100 Mbit/s Ethernet interface ports. SMB-2000 generates, monitors and captures Ethernet traffic in half or full duplex modes.

Each interface card is controlled by an application which provides testing facilities such as throughput, latency, packet loss and back-to-back performance measurements.

A throughput test starts with generating traffic at wire speed, e.g. maximum transmission rate. If any packets are dropped, it decreases the load to 80% of the previous rate and runs another trail again. The subsequent trails are determined by a binary search to find the half way between the last successful rate and the last failed rate. The binary search continues

until it reaches a predefined resolution. The throughput is the maximum rate found without any packet loss.

A latency test generates a traffic load for a specified period of time. Half way through the test, the latency of one packet is measured. A latency is measured from the time the last bit is sent on the output port to the time the first bit is received at the input port.

A packet loss test generates a heavy traffic load for a period of time. At the end of the test, a report is generated which specifies the number of packets dropped.

A back-to-back performance test generates a burst of traffic at maximum frame rate and measures the burst size in number of packets. If no packets are dropped, the burst size is increased and the test is repeated. If packets are dropped, the burst size is decreased by 50% and the test is repeated. At the end of the test, a maximum burst size at which no packets are dropped is presented. The presented burst size is the buffer capacity of the device under test.

In addition to the above predefined tests, an individual test setup is also possible where each packet stream characteristics can be adjusted, such as transmission rate, packet length, protocol or port number, to achieve the expected simulated traffic mixture.

### 3.2.2 A Network Management Application

We wanted to collect information from network elements in an as easily and economically feasible way as possible. The purpose of this testbed is to simulate a real network which can be geographically scattered and any major impact on the network is highly unwanted. This means that we are looking for a reasonably cheap tool, which uses existing infrastructure in a standardized way with minimal impact on the network. The Scotty software package [2] fulfills all of the above needed requirements. Scotty uses among other things, SNMP for transferring information to/from network elements and is also available from the Internet free of charge.

Scotty [2] is a software package which allowing implementation of site specific network management software using high-level, string-based APIs. The software is based on the Tool Command Language (Tcl) which simplifies the development of portable network management scripts. Scotty consists of two major components. The first one is the Tnm Tcl Extensions [4] which provides access to network management information sources. The second component is the Tkined network editor which is a programmable graphical user interface (GUI) for network management applications. It allows the user to draw and maintain a map of a network configuration. Objects on the map can be selected and manipulated by either built-in tools or external scripts that communicate with the Tkined editor. The first component, the Tnm Tcl Extensions suits our purpose well to collect information from network elements and will be further explained.

Tcl is an interpreted high-level scripting language, although Tcl compilers are available if needed. Tcl is highly portable and extensible, it runs on almost every platform. Tcl extensions are made in the C programming language or in Tcl itself in an easy and convenient way. Scotty is an example on extensions made in C and is based on the event-driven pro-

gramming paradigm found in Tcl. A typical Scotty application loads an initialization script which installs some basic event handlers. When the initialization is completed it enters an event loop. Tcl scripts are evaluated to process events that are created when a message is received from the network or when a timer expires. The event loop terminates if no timer events are left or no more messages are accepted from the network.

The primary goal of the Tnm Tcl Extensions is to provide a string-based API to access network protocols and services relevant for network management applications. Some of the protocols and services supported by Tnm extensions are DNS, HTTP and SNMP. An example of a Tnm extensions script is ciscoQueue.tcl found in Appendix A.2. ciscoQueue.tcl collects output queue information periodically or on demand from a Cisco router, and stores the retrieved information locally in a file. This example, shows how easy an access to network management functions implemented in C can be. The script follows the Tcl conventions described in [11].

Another very useful application that is also part of the Scotty software package is the graphical SNMP MIB browser. SNMP MIB browser makes it possible both to explore the MIB tree structure with enclosed MIB definitions and to retrieve current information from management agents installed in a network via SNMP.

### 3.2.3 A Delay Measurement Tool

Measurable One-way Observer (moo) [5] is a software application for end-to-end delay measurement. Moo makes one-way delay measurements as experienced by a probe packet that travels across an IP network from the source host to the destination host.

To be able to make accurate one-way delay measurements, two things must be fulfilled: First, a common reference clock for all involved measurement hosts must be realized. The second thing is to reliably timestamp the packets when they were transmitted and received.

A common reference clock is achieved by using the Global Positioning System (GPS) which provides Universal Coordinated Time (UTC) information. Each measurement host must have a GPS device installed which receives the UTC timing information from the GPS system. The GPS driver, which controls the GPS device, steers the hosts system time towards UTC. The hosts system time reaches the UTC time within 1 hour, with an accuracy of 0.1 ms.

A common reference clock can also be obtained by a separate timing channel between measurement hosts. The GPS device has the ability to operate in a number of different modes. It can operate in a *generator mode* where the system clock works as a reference clock. In the *generator mode*, the timing information can be distributed to other hosts. In the *synchronized generator mode* operates the device towards synchronization via an external time reference. The last operative mode is the *GPS synchronized mode* which was explained above. The same GPS device and driver can operate in different modes with minor configuration changes. The synchronization of connected hosts is achieved by transmitting a synchronization pulse every second. The connection between the synchronized hosts are achieved by an ordinary pulse code modulation (PCM) cable.

The moo software timestamps probe packets which are traversing the network. Moo can run both in transmit and receive mode. A sending host, which runs in transmit mode, timestamps a probe packet with the local system time, i.e. the UTC time, and sends the packet to the destination host. When the packet reaches the destination host, which runs in receive mode, the arrival time is logged with the destination host's system time. If the two hosts are synchronized, the difference between the send and the receive timestamps is the measured delay for the packet, and thus, the delay through the network.

With a synchronized system as above, a measurement accuracy of 0.5 milliseconds can be achieved.

## 3.3 *A Working Testbed*

The purpose of the testbed has been to implement parts of a DiffServ network and be able to collect information, as conveniently as possible, from network elements and external equipment. Network elements, i.e. DS nodes, should supply both network related information and information about the node itself. External equipment should assist with end-to-end delay measurement information. A similar work has been done in [9] but on a much larger scale and with more DiffServ functionality. Our aim was to implement the inner part of a DS domain, i.e. functionality which can be found in an interior DS node, such as packet classification and a mechanism for supporting PHBs. The testbed was also going to support a service for real-time traffic. The only service in DiffServ suitable for real-time traffic is Premium Service. Since Premium Service will not be the only service provided by an internet, at least some other service will also be provided [8]. In the Internet today, best-effort service is the only available service, and will probably be part of the Internet in the future. Therefore, the testbed should have DiffServ implemented with both Premium and best-effort services. We start by describing how to implement Premium Service in the testbed router, best-effort service is implemented by default. We continue with describing how to configure the equipment for the measurements and how this affects the configuration of the routers. But first, a short introduction to Premium Service.

Premium Service [8], [9] is mainly a service of commercial interest and should not replace best-effort service in any way, but rather coexist in the same network. It is economically desirable to provide several services in the same network. Each flow or aggregation of flows should have a characteristic of almost no loss, no delay and no jitter. Premium Service could be compared with a virtual leased line or a standard telephone line.

Premium Service had to be implemented by one of the four available scheduling methods implemented in the router, as described in Section 3.1.1. Due to the Premium Service real-time characteristics and the small delay budget available in a mobile IP network, see Section 1.2, we were looking for a scheduling scheme which provides a minimal delay for real-time traffic.

The first scheme, FIFO, can not distinguish between different kinds of traffic and therefore is not suitable for real-time traffic. The second scheme, PQ, can divide the traffic into several queues and the queue with highest priority can provide real-time characteristics. The third scheme, CQ, is based on a round-robin scheme. A Round-robin behavior will

introduce additional delay on all output queues, even the one with real-time traffic which is not what we are looking for. Another drawback with CQ is that it must be configured for a specific amount of real-time traffic, this is not done automatically. If the prerequisites change, CQ must be reconfigured. The last and the fourth scheme, WFQ, can differentiate traffic but the prioritization is not changeable. This means, there exists only one way to prioritize among the traffic streams. High priority traffic has already a predefined relative priority over low priority traffic which do not suit our purpose the way we intended. WFQ does not provide a minimal delay with its fair queueing algorithm. Additional delay will always appear for high priority traffic when low priority traffic is present. This lack of functionality makes it difficult to provide real-time services. Based on the above discussion, PQ was chosen as scheduling scheme for implementing Premium Service. The selection of PQ is also in accordance with what was recommended in 4.2.2.4 in [7].

Premium Service was realized in the testbed routers by PQ based on the precedence bits in the IP header. A similar configuration for another type of router was made in [10]. Mapping of precedence values to a certain per-hop-behavior is described in 4.2.2.2 in [7]. This description fits well into the relative order between the priority queues in the PQ scheme. A mapping between a packets precedence values and Cisco's PQ was done by connecting access-lists to certain priority queues, and each access-list connects to a specific precedence value. A more thorough description can be found in Appendix A.3. The mapping is shown in Table 3-1.

***Table 3-1** Mapping between IP Precedence values and Priority queues*

| IP Precedence values | Priority queue |
|---|---|
| 6 - 7 | high |
| 4 - 5 | medium |
| 2 - 3 | normal |
| 0 - 1 | low |

With this mapping, two precedence values will use one priority queue. This is not according to the requirements in 4.2.2.2 in [7] unfortunately, but every second precedence value has the required behavior. This priority scheme has strict priority, which means all excess traffic will be discarded and not prioritized to a lower level. If there exist packets in a higher priority queue, this queue will first be scheduled for outgoing traffic and if a lower priority queue fills up to the maximum queue size, any excess traffic to that queue will be discarded.

End-to-end delay measurements were made by the measurement tool moo. Moo was implemented in each end-host and these hosts were also synchronized. A common reference clock can be achieved by the GPS hardware in two ways. The first way is the scalable approach, were GPS hardware is connected to a GPS antenna, which means that the geographical location is of no importance. The second approach is simply to connect a PCM cable between the hosts. The latter solution is obviously very limited, but in a small network like this testbed, it is feasible. For this testbed, there were no GPS antennas available, so the synchronization was achieved by a PCM connection between the two hosts (the dotted line

in Figure 3-3). We wanted to measure flows of probe packets with both premium and best-effort services. We had only two hosts for the delay measurements available, and neither the moo software nor the hosts operating system had the ability to have several different precedence bit settings concurrently, which was needed for the measurements. This problem was solved in the router software by access lists. An access list could identify a probe packet, by a UDP port number, and then use CAR to reset the precedence bit before the packet was scheduled for an outgoing interface. By using different UDP port numbers, several different precedence bits could be set.

Collection of network information was done by using the Tnm part of the Scotty package. A number of Scotty scripts were made to retrieve the requested information. A similar work with Scotty scripts has been done in [12]. Each script, working as a SNMP manager, sends a SNMP request to the relevant network element. The network element, which must support an SNMP agent, replies to the manager with the requested information. All retrieved information is stored locally in the manager with date, time and name of the network element. The name of the network element is retrieved from a name server configured in the testbed. To be able to keep track of the delay measurements and the measurements retrieved from the network elements, both the moo hosts and the SNMP manager must be synchronized. This was done by installing GPS hardware in the SNMP manager and connect it to the PCM link, thus all three hosts are using the same reference clock.

The testbed must be able to generate traffic. The traffic generator [3] has three Ethernet ports, each port can be configured for both half or full duplex traffic. One port could be configured for receiving all traffic or streams from the other ports. This configuration is useful if packet loss or delay measurements are of interest. See Figure 3-3 for the final result.

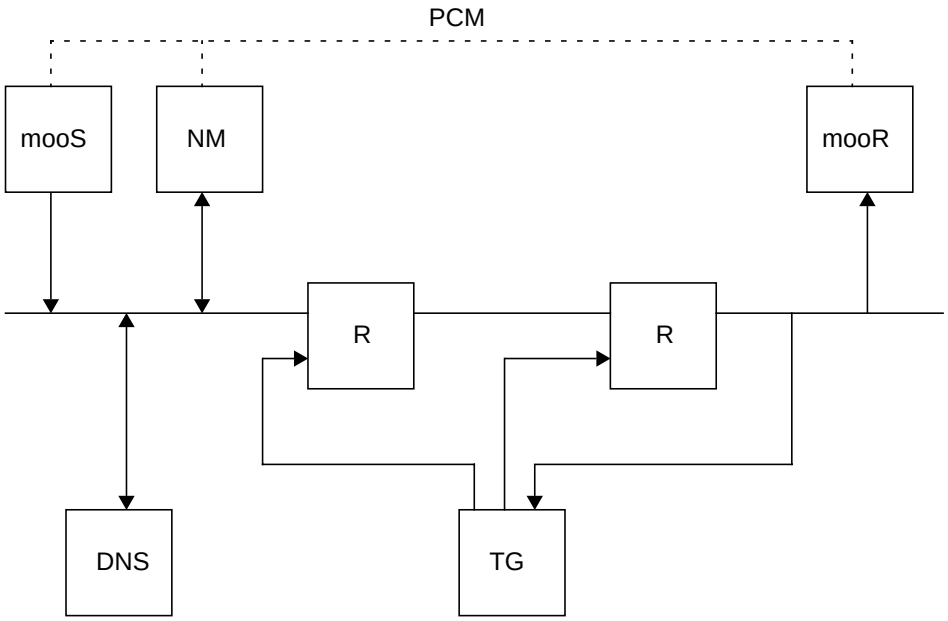


***Figure 3-3 The configuration of the testbed****. A working testbed with two routers (R), a traffic generator (TG), a name server (DNS), a NMS manager (NM) and two hosts for the one-way delay measurement tool, mooS and mooR for sending respectively receiving probe traffic.*

# 4 PROBLEMS IN PREDICTING AVAILABLE CAPACITY

Internet is in a transition from a one-service network to a commercial multi-service network. In a multi-service network, the network capacity for the different providers will be bought and sold among the providers to fulfill their customers' demands. To be able to trade network capacity, it must be predicted in some way. This chapter will make an effort to predict the capacity in each service class in a DiffServ network. One possible way to predict the capacity in a network, is to predict the capacity for each network element. A collection of measurements from all network elements can then be used by learning methods to estimate the whole network's capacity or even predict the load of the network in the near future.

We wanted to predict the capacity for each service class in a router. The router was configured for Premium Service (PS) and Best Effort service (BE). All measurements were collected via SNMP and via the traffic generator, see Section 3.3 for a more thorough description of the configuration. The traffic was constrained to only UDP packets due to limitations in the traffic generator.

## 4.1 *Test Traffic*

Traffic was injected into the router via two interfaces and concentrated towards the third interface (output port), see Figure 4-1. The traffic was divided equally between the two interfaces, e.g. each interface has 21% PS and 79% BE service traffic. The traffic was generated in a steady stream with no bursts. This is a rather simple traffic characteristic and distribution. The reason was to simplify the process of finding methods to predict the capacity.

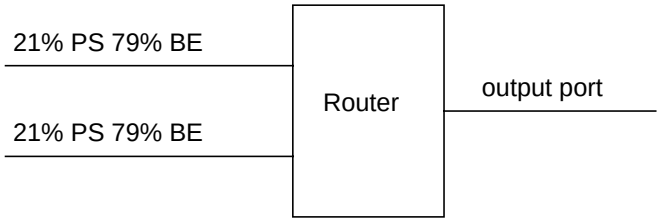


***Figure 4-1 The traffic distribution through the router****. The offered load is equally distributed among the two input ports. All measurements were made on the output port.*

Best Effort service is mainly used to carry traffic consisting of file transfers i.e. FTP, HTTP, e-mail etc. thus we used a packet size of maximum transfer unit (MTU) to minimize the overhead. A MTU for Ethernet is 1500 bytes which was also used as a guideline to simulate best-effort traffic. The Ethernet frame was set to 1500 bytes so the actually IP packet size was 18 bytes shorter i.e. without MAC header and FCS.

Premium Service is for real-time traffic which uses a smaller packet size. For instance, a speech channel in GSM uses a bandwidth of 13 kbit/s. Every packet contains 20 ms of speech, i.e. at 8000 samples/s, which gives a payload of 32.5 bytes. With IP and UDP

headers added, it will result in an IP packet size of approximately 60 bytes. We selected the GSM packet size to simulate PS traffic, i.e. an Ethernet frame size of 80 bytes with included MAC header and FCS.

The following sections, cover different methods used to predict the capacity and why the methods were chosen.

## 4.2 *Queue Monitoring*

We wanted to measure the amount of traffic for each service class, i.e. PS and BE. In the router, different queues are used to differentiate PS and BE traffic. One possible method to predict the capacity, is to monitor every queue used by the service classes in the router.

Implementation of queues in routers differs substantially between router vendors and models. Almost every model needs its own set of queue MIB variables. We used a Cisco enterprise queue MIB (CISCO-QUEUE-MIB) which contains a number of variables for accessing information about the queues, e.g. maximum queue length, current queue length and number of packets dropped for each queue.

We started to load the router by generating traffic at 500 kbit/s, i.e. 250 kbit/s per incoming interface. Each trial ran for 2 minutes. The traffic was slowly increased, and at the same time, the queues were continuously monitored. At a load of 4.3 Mbit/s (the dotted vertical line in Figure 4-2) something happened. The router stopped responding to any queries via SNMP and thus no queue information was available. It even stopped responding to any communication via a directly attached console. It could be two reasons why the router was not responding to our queries. The router was either bandwidth or processor capacity limited. This occurred at 4.3 Mbit/s on a 10 Mbit/s Ethernet connection so it can not be bandwidth limited.

Figure 4-2 shows a comparison of throughput between Priority and FIFO scheduling. At 4.5 Mbit/s, PRIO starts to degrade rapidly in capacity while FIFO continues to carry traffic up to 9.2 Mbit/s. With Priority Queueing enabled, the router should be able to carry traffic in the same degree as FIFO, but is limited by the processor capacity. It seems it can not handle more traffic than 1750 packet/s (4.5 Mbit/s) and above. Measurements above 4.5 Mbit/s were made by the traffic generator. The figures for offered/carried load includes framing characters, i.e. MAC header and FCS, which gives a better utilization degree than the actually transmitted Ethernet payload.

Another thing worth to be noted in Figure 4-2, is how the router handles overload situations. It is remarkable how different the router handles the overload for FIFO and PRIO Queueing. With FIFO enabled, the router shows a stable behavior even when the offered load is twice the carried load. But with PRIO enabled, the router starts rejecting traffic at 45% bandwidth utilization and almost stops carry traffic (0.3 kbit/s) at 65% utilization. This is a hazardously behavior were the router seems to be working fine at 40% utilization and then is almost dead at 65%.

We tried to measure the amount of traffic for each service class by monitoring queues in the router. This was not possible. We could not retrieve any queue information above a load of

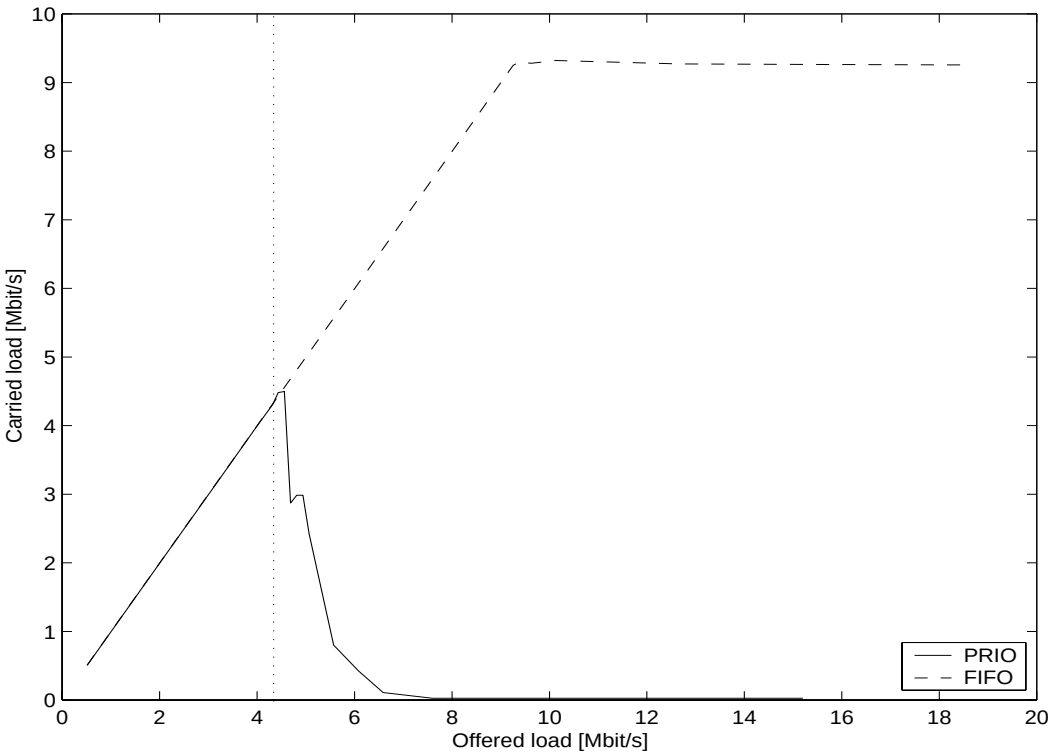


***Figure 4-2 Comparison of throughput between Priority and FIFO scheduling****. The Figure shows how the router, with PRIO queueing enabled, with an offered load of 4.5 Mbit/s drops to 0.3 kbit/s while FIFO queueing continues up to 9.2 Mbit/s and keeps the carried load in a stable way.*

4.3 Mbit/s which was caused by limitations in processor capacity of the router. The problem with the routers processor capacity and its unstable behavior raised another question. Is it possible to predict the peak performance in the router with PRIO enabled? This will be covered in the next section.

Before we continue with the next section, a slightly different performance test was made to explore how severe the processor capacity limitation was with PRIO enabled and how it behaved in an overload situation.

We wanted to measure how Priority Queueing affected the performance of the router regarding throughput without any external disturbances as possible. The offered load was changed to only consist of 80 bytes BE packets which was injected into the router via one incoming interface. Only one interface was used for incoming traffic due to reduce disturbances caused by external equipment and the shared media, i.e. collisions on the Ethernet. Two tests were performed, one with PRIO and one with FIFO enabled. See Figure 4-3.

Figure 4-3 shows how the router, with FIFO scheduling enabled, carries all offered load up to a maximum of 12019 packet/s which is also the maximum utilization of the media. With PRIO scheduling enabled, the router only carried traffic up to 2125 packet/s. After the maximum at 2125 packet/s, the carried load decreases rapidly down to 89 packet/s at an offered load of 3000 packet/s. The router keeps the carried load of 89 packet/s even when

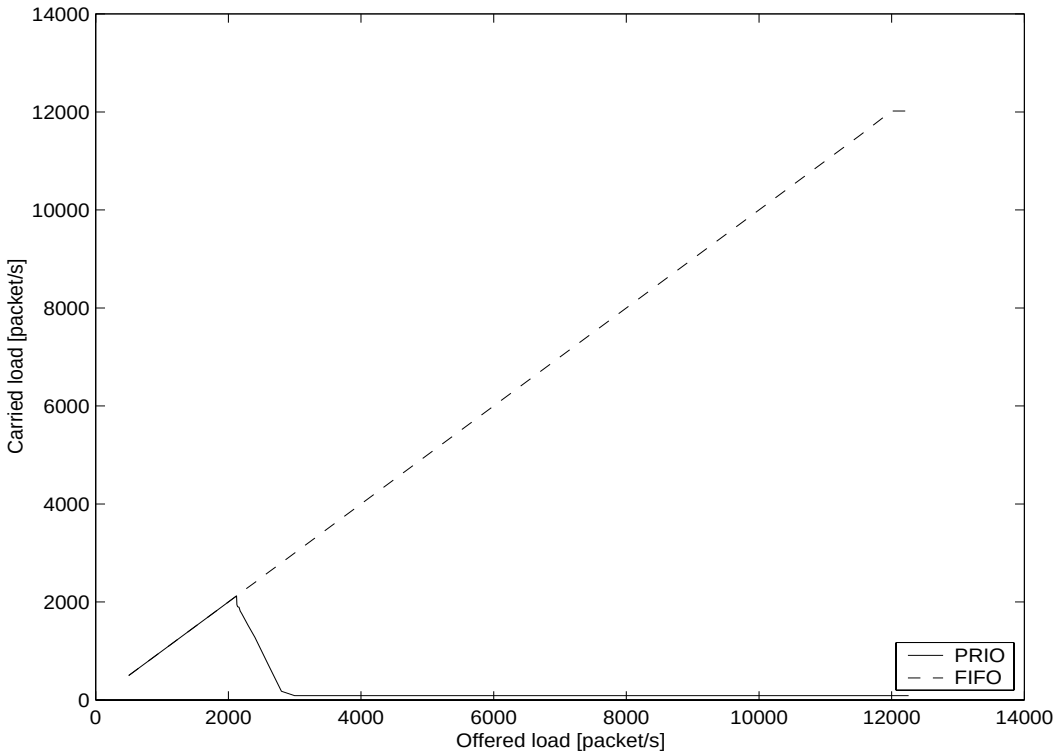


***Figure 4-3 Comparison of throughput between Priority and FIFO scheduling****. The Figure shows how the router with PRIO queueing enabled reaches the maximum at 2125 packet/s and then drops to 89 packet/s while FIFO queueing continues up to 12019 packet/s which is the maximum utilization of the media.*

the offered load is increased to its maximum at 12019 packet/s. 89 packet/s seems to be the minimum carried load at an “overload” situation.

With Priority Queueing enabled, the router can only perform 18% of the maximum at 12019 packet/s which is remarkable. This is a performance degradation of 82%. The only difference from the FIFO trial was the enabling of PRIO. No CAR or other capacity demanding functions were used.

## 4.3 *Processor Load*

The problem with the behavior of the router depicted in Figure 4-2, made us to investigate if it was possible to predict the processor load before the carried load drop to a minimum (0.3 kbit/s).

We measured the overall processor load and also tried to identify internal processes which could relate to different types of traffic. We used a Cisco enterprise processor MIB (CISCO-PROCESS-MIB) for retrieving the overall CPU busy percentage in the last 5 second period. It was not possible to identify any CPU processes, nor any process related MIB variables, which could be used for our purposes the way we intended. Figure 4-4 depicts the relation between offered load and CPU utilization.

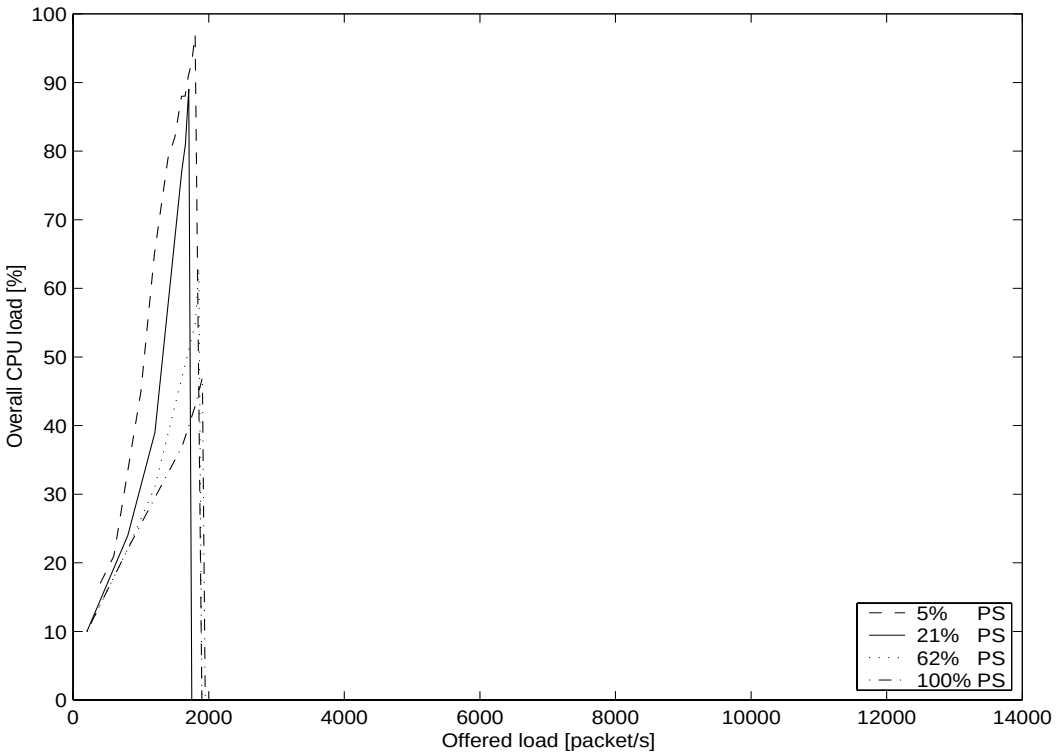


***Figure 4-4 Relation between offered load in packet/s and CPU utilization for different PS distributions****. After each peak, there is a drop which indicates where the router stops responding, i.e. the maximum CPU load. The four different PS distribution lines show a concentration near 2000 packet/s where the maximum carried load seems to be regardless of the amount of PS traffic offered.*

The Figure shows a number of runs with different PS and BE distributions. The drop after each peak indicates where the router stopped responding and thus no more CPU information was available. The drop for each line occurs in a narrow range in number of packet/s (1750-1950 packet/s). This concentration near 2000 packet/s is regardless of PS distribution and thus probably indicates the area where the maximum load in packet/s must be for Priority Queueing.

Figure 4-5 shows a similar relation but with offered load in Mbit/s. The peak for each PS distribution is distributed along the offered load axis but with different CPU load. This is very strange. How can it be a difference in CPU load of approximately 50% when measure the same behavior, i.e. an overloaded processor, for each PS distribution? What can also be seen is a decrease in CPU load when the amount of PS traffic increase, i.e. when the amount of short packets increase. When the amount of short packet increase, a lower bandwidth utilization occurs, i.e. caused by the 2000 packet/s limitation. This indicates a relation between the measured CPU load and the bandwidth utilization. A low bandwidth utilization will give a low measured CPU load and vice versa. Obviously, short packets cause problems for the router which also affects the CPU information and makes the information unreliable.

With Priority Queueing, we can not measure when the processor limit is reached. But we have two other scheduling schemes worth investigating, namely Weighted Fair Queueing

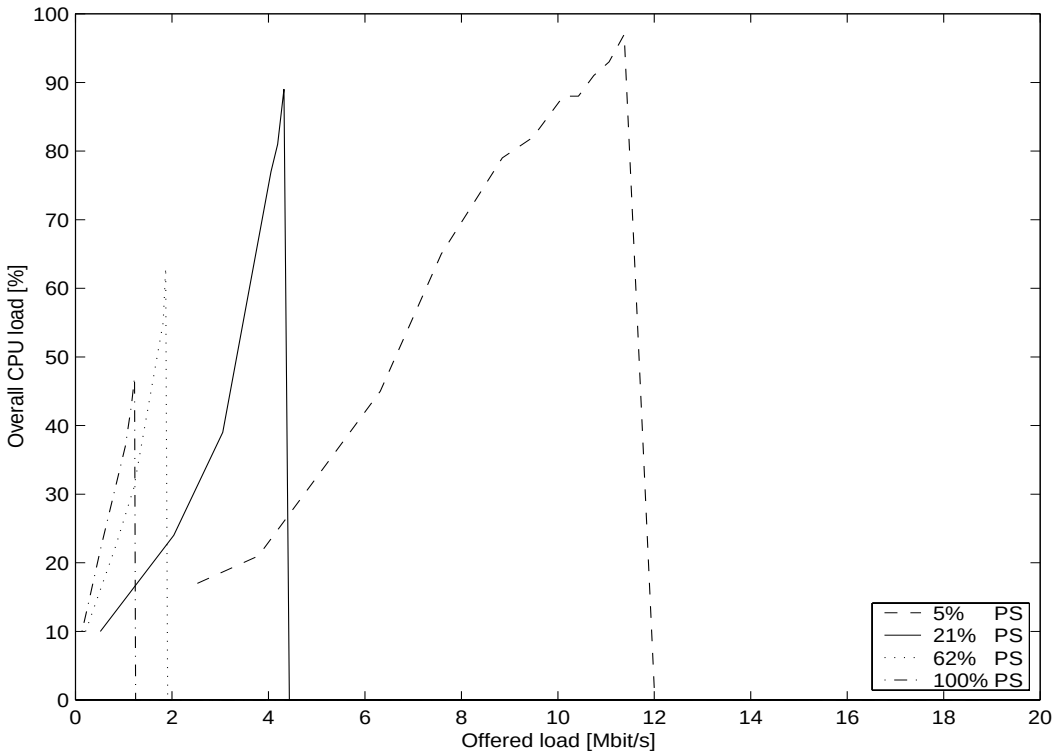


***Figure 4-5 Relation between offered load and CPU utilization for different PS distributions****. After each peak, there is a drop which indicates where the router stops responding, i.e. the maximum CPU load. The four different PS distribution lines show that the maximum CPU load decreases when the amount of PS traffic increases.*

(WFQ) and Custom Queueing (CQ). These two scheme were not selected to provide real-time services because of the additional delay they introduce. Nevertheless, they are worth to be investigated for their ability to differentiate traffic. We start with the same tests for WFQ and then continue with CQ to see if we can present other results than with PRIO.

Figure 4-6 depicts a number of runs with WFQ enabled. The drop for each PS distribution occurs in the range of 2200-2800 number of packet/s. This result shows a slightly better performance than PRIO but far from what it should be able to perform.

Figure 4-7 shows the relation between offered load in Mbit/s and CPU utilization for WFQ. WFQ has the same problem as PRIO to present the accurate CPU load. The CPU load decreases as the PS distribution increases, similar as PRIO but with little more accurate figures.

Figure 4-8 presents the drops for CQ. They are in same range (1800-2400 packet/s) and thus also shows almost identical bad performance as the other schemes.

Figure 4-9 depicts the CPU load for CQ. This is identical to PRIO and can not be used for predicting the processor overload.

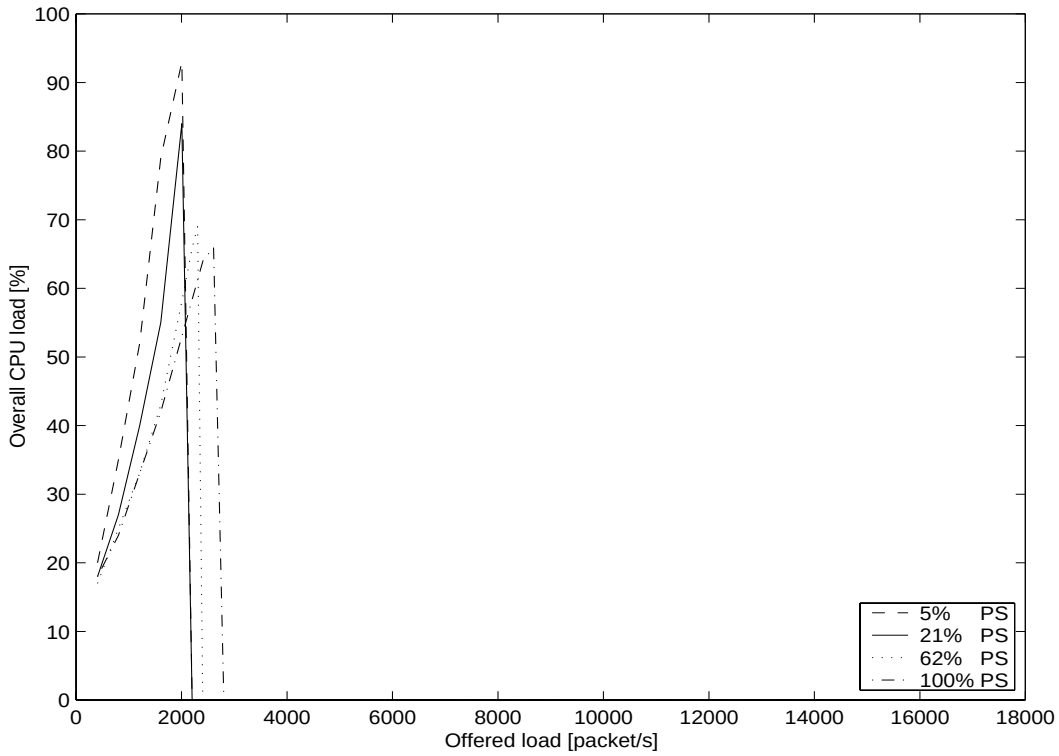


***Figure 4-6 Relation between offered load in packet/s and CPU utilization for WFQ**. The drops occurs in the range of 2200-2800 number of packet/s which is slightly better than PRIO.*

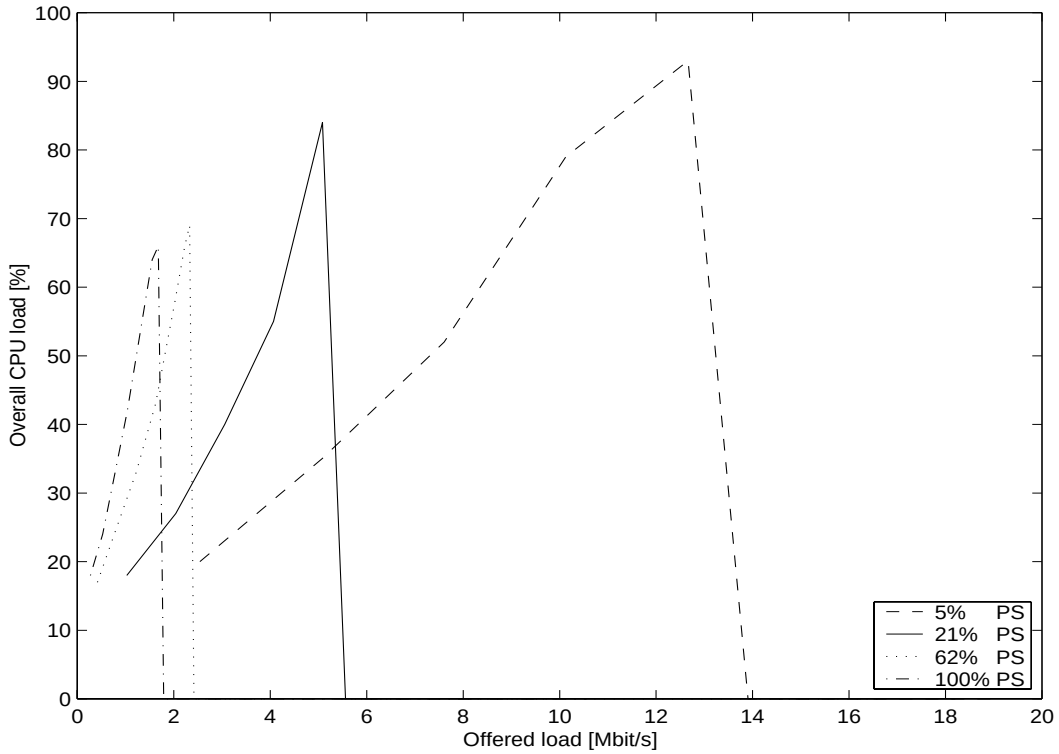


***Figure 4-7 Relation between offered load in Mbit/s and CPU utilization for WFQ.** WFQ presents similar inaccurate CPU load as PRIO.*

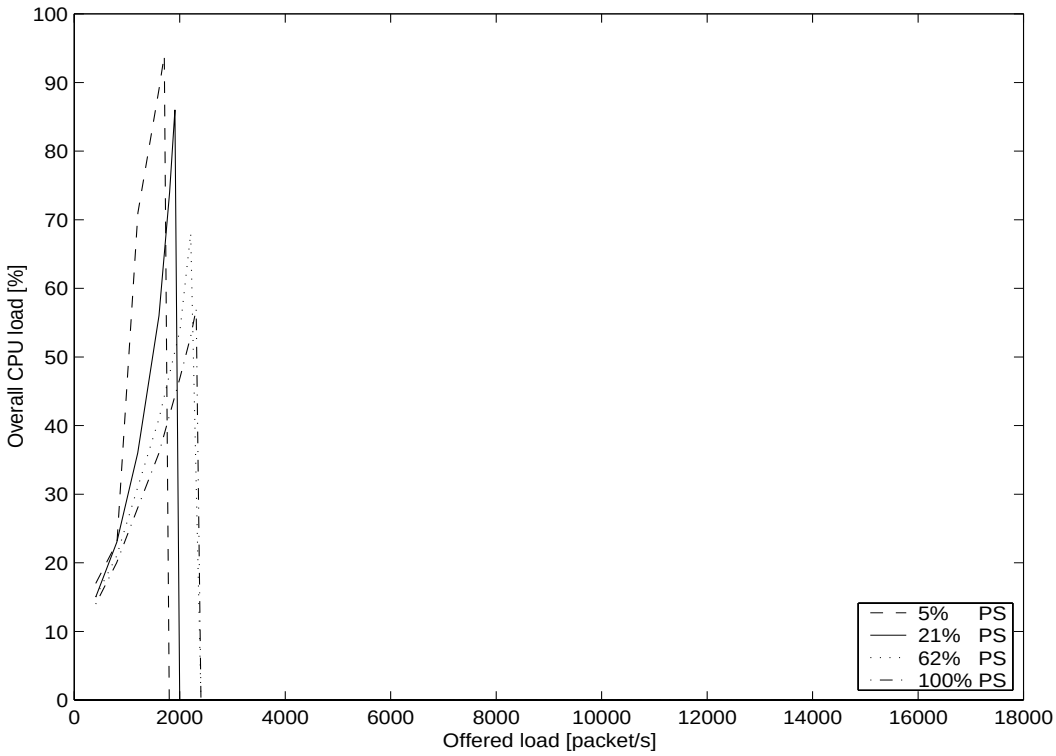


***Figure 4-8 Relation between offered load in packet/s and CPU utilization for CQ**. The drops occurs in the range of 1800-2400 packet/s which is also slightly better than PRIO.*

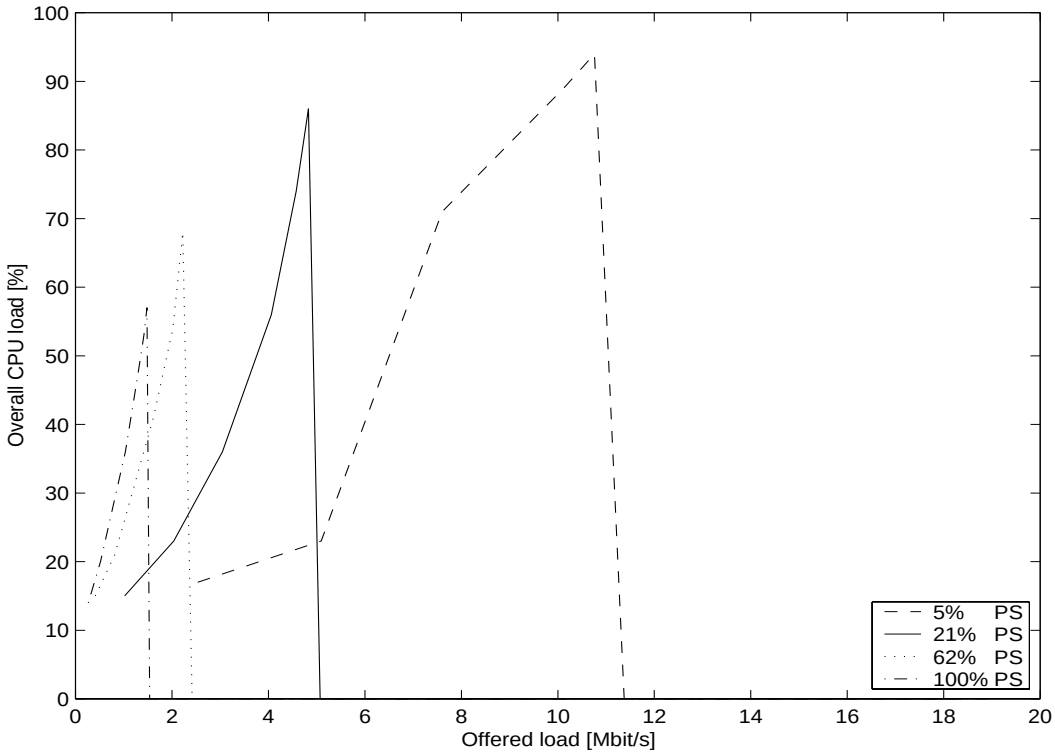


***Figure 4-9 Relation between offered load in Mbit/s and CPU utilization for CQ.** CQ presents almost identical CPU load as PRIO.*

All measurements made on WFQ and CQ show similar or almost identical characteristics as PRIO, and thus, neither of these two scheduling schemes can be used to predict the limitation of the processor.

We conclude the CPU load measurements by presenting all available scheduling schemes and their performance in Figure 4-10. Two interfaces were used for incoming traffic compared to Figure 4-3. The figure shows a stable behavior of FIFO while the other scheduling schemes get overloaded very quickly (17-22% of maximum) and then drop to less than 1% of maximum. However, WFQ gives the best performance of the investigated scheduling schemes but can only provide 22% capacity of maximum which is very poor. The most important aspect of these result is how every investigated scheme behave in an overloaded situation. It is not an acceptable behavior when a router stops carrying traffic on a particular offered load, and which also is not measurable. A router with this type of characteristics will probably not be deployed with real-time traffic.

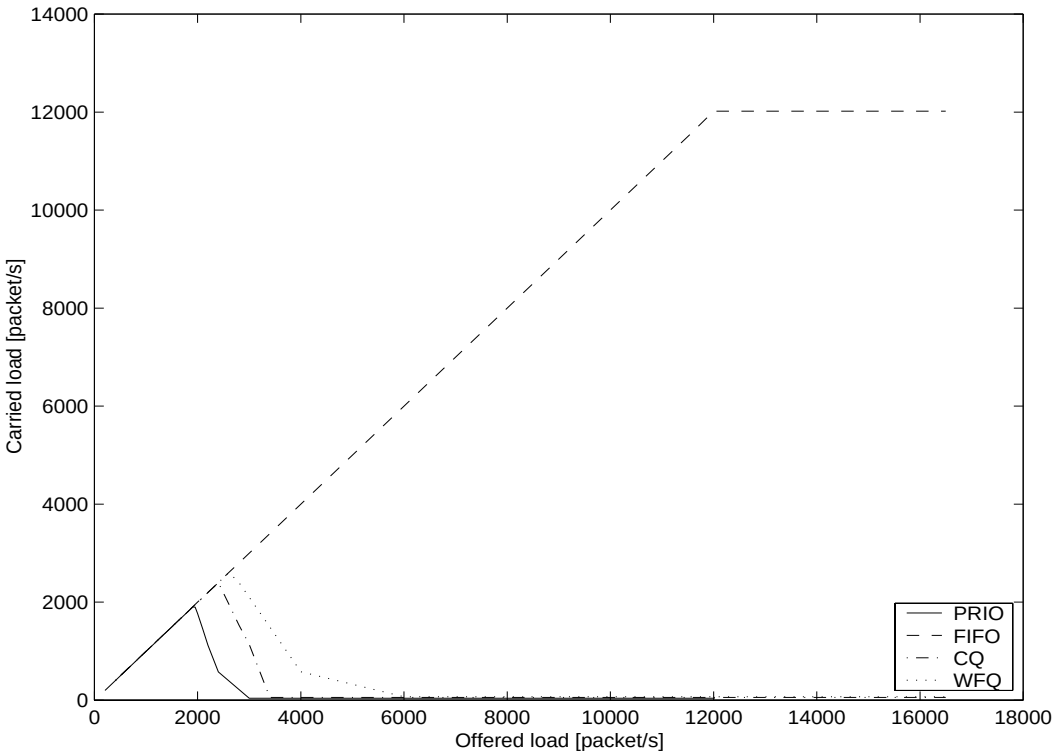


***Figure 4-10 Comparison of throughput for FIFO, PRIO, CQ and WFQ.*** *All scheduling schemes which can differentiate traffic show very poor performance. WFQ provides best performance with 22% of maximum capacity*

The CPU load of the router for every scheduling scheme except FIFO gives misleading information and thus is not useful for our intention. We continue to investigate if the Ethernet interface and the IP MIBs can give us what we want. These MIBs are not affected by the scheduling scheme and thus will be covered in the next section.

## 4.4 *IP and Interface Information*

Each Ethernet interface has a number of counters implemented according to standard interface MIBs [14],[15]. They specify among other things, e.g. the supported MTU size, the current operational speed etc. The router also supports IP MIBs [16], [17] which specifies, e.g. number of IP datagrams received, number of IP datagrams forwarded etc.

We tried to find ways to differentiate traffic, i.e. by IP header information, and to measure how traffic flows in and out from a router, e.g. see if the traffic is symmetric or concentrated in some direction.

Information about how traffic flows through the router was easy to collect. All necessary MIB counters were already available, e.g. number of bytes and packets sent and received on an interface.

Information about the IP header could not be retrieved. There were no MIB variables available for that kind of information.

The standard interface MIBs can be used to predict the remaining capacity, but only for the aggregation of all traffic streams. All traffic is mixed at the interface and there are no ways to extract information about different types of traffic. Everything is a mixture of packets and bytes which is only applicable to one service class.

The last attempt to find any methods for our purpose is to specify critical levels internally in the router which the next section will cover.

## 4.5 *Alarm Setting*

Another solution could be to set alarm internally in a router. When a particular status is reached in a router, e.g. processor load is 80%, an alarm is issued. This method has a rather bad resolution but could be of interest if certain critical alarm levels are identified.

In Cisco's IOS software, alarms can be specified in several ways. Alarm thresholds can be set independently when they are issued and ceased, e.g. issue an alarm at 80% processor load and cease the alarm at 60%. Alarm notifications are realized by sending SNMP *traps* to a network manager.

Alarms can be specified by predefined *traps* or MIB variables. There exists only a few predefined *traps*. They are either public SNMP *traps* or Cisco enterprise SNMP *traps*. Both of these *traps* are related to link states, e.g. link is up, and the operation of a router, e.g. a reload has occurred. This type of information is not very useful for us. A MIB specified alarm can use any available MIB variable. The only variables of interest were already covered in the previous sections and thus not very useful for us.

# 5 CONCLUSIONS

The objective of this master thesis was twofold:

- to build a multi-service testbed based on commercial products, and
- to provide means to predict the capacity of routers.

A testbed has been built to support a multi-service network and additional test equipment has been added to the testbed for measurement purposes. Commercial routers have the ability to differentiate traffic but in a limited way. In present routers, there are only a few number of possible ways to differentiate traffic into service classes and treat the traffic accordingly to the expected service. But indications from router vendors show that these possibilities extend in the near future. New software releases support more functionality for real-time services and this will increase, according to the demands, for such services.

Attempts to predict the capacity in the network were made by principally four methods, i.e. queue monitoring, processor load, interface and IP information and finally by alarm setting. They showed different kinds of limitations, and thus, none of these four methods could be used for our purpose the way we intended. The possibilities for measurements were intended for only one service class, and thus, could not help us to measure the capacity for several service classes.

However, the most remarkable result must be the limitation in processor capacity. By only enabling Priority Queueing, the bandwidth capacity of the router decreased with 84% (at 100% PS traffic) and it also showed a very unstable behavior of the limitation. A small increase in traffic load, caused the router from carrying all offered traffic to rejecting almost all traffic. These results were not only valid for Priority Queueing, all scheduling schemes which could differentiate traffic showed similar behavior.

We have shown the difficulty in measuring existing office routers for real-time traffic. New routers must be design for measurability to be able to provide necessary information for capacity prediction.

Do not believe what vendor claims. They state wire-speed for their products, but this is not entirely true. Wire-speed can be achieved with a special configuration but this in not valid for all supported functionality of the product.

If real-time traffic is required, measure with your configuration and your packet sizes. You have to know what you put in, unless routers are wire-speed at all supported functionality.

Old and new equipment must be treated differently. Old equipment will probably be a diversity of software and hardware versions, and thus, all old equipment must be performance tested before real-time services can be put into practise.

## 5.1 *Suggestions for Further Studies*

We failed to provide means to predict the capacity for several service classes in a network. This area needs further investigation. Some interesting topics for further studies in this area are listed below:

- This thesis was limited to only one router model with associated software. New software releases for this model are coming up which better support real-time services. Investigation is needed if the new software can be used to predict capacity.
- Use other vendors' products to see if they have similar limitations that we discovered or if they have some better ways to predict capacity.
- A new MIB draft for the Differentiated Services [18] has been proposed. This draft could be part of the next step for further studies in this area.
- To be able to better simulate a real network, the testbed can be expanded with more routers so that more complicated traffic cases can be explored. Naturally, this depends on the ability to provide necessary information from the routers.

# 6 REFERENCES

[1] Cisco Connection Online by Cisco Systems, Inc. available from http://www.cisco.com/, Mars 2000

[2] Scotty - Tcl Extensions for Network Management, available from http://wwwhome.cs.utwente.nl/~schoenw/scotty/, Mars 2000

[3] Netcom Systems Home Page, available from http://www.netcomsystems.com/, Mars 2000

[4] J. Schönwälder, H. Langendörfer, “Tcl Extensions for Network Management Applications”, Tcl/Tk workshop, July 1995

[5] V. Ayadurai, “Tool for Accurate One-way Packet Delay Measurements”, MSc Thesis Project - SwitchLab, Ericsson Radio Systems AB, May 1999

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, “An Architecture for Differentiated Services”, RFC 2475, December 1998

[7] F. Baker, D. Black, S. Blake, K. Nichols, “Definitions of the Differentiated Services Field (DSFIELD) in the IPv4 and IPv6 Headers”, RFC 2474, December 1998

[8] K. Nichols, V. Jacobson, L. Zhang, “A Two-bit Differentiated Services Architecture for the Internet”, RFC 2638, July 1999

[9] B. Teitelbaum, S. Hares, L. Dunn, R. Neilson, V. Narayan, F. Reichmeyer, “Internet2 QBone: Building a Testbed for Differentiated Services”, IEEE Network, September/October 1999

[10] O. Hagsand, K. Hansson, “Real-time IP Delivery of Interactive Audio”, Ericsson Business Communications, January 1998

[11] R. Johnson, “Tcl Style Guide”, August 1997

[12] A. Olah, “Analysis of Delay Measurements”, Traffic Lab, Ericsson Hungary, December 1998

[13] W.R. Stevens, ”TCP/IP Illustrated Volume 1 The Protocols”, Addison Wesley, July 1997

[14] K. McCloghrie, F. Kastenholz, “The Interfaces Group MIB using SMIv2”, RFC2233, November 1997

[15] J. Flick, J. Johnson, “Definitions of Managed Objects for the Ethernet-like Interface Types”, RFC2665, August 1999

[16] K. McCloghrie, M. Rose, “Management Information Base for Network Management of TCP/IP-based internets: MIB-II”, RFC1213, March 1991

[17] K. McCloghrie, “SNMPv2 Management Information Base for the Internet Protocol using SMIv2”, RFC2011, November 1996

[18] F. Baker, K. Chan, A. Smith, “Management Information Base for the Differentiated Services Architecture”, draft-ietf-diffserv-mib-02.txt, March 2000

[19] T. Kushida, “An empirical study of the characteristics of Internet traffic”, Computer Communications 22:1607-1618, 1999

[20] V. Paxson, “Towards a Framework for Defining Internet Performance Metrics”, Proc. INET96, 1996

[21] V. Paxson, “End-to-End Internet Packet Dynamics”, Proc. of ACM SIGCOM’97 pp. 139-152, 1997

[22] K.C. Claffy, G.C. Polyzos, H.-W. Braun, “Application of Sampling Methodologies to Network Traffic Characterization”, Proc. ACM SIGCOM'93 pp. 194-202, 1993

[23] V. Paxson, J. Mahdavi, A. Adams, M. Mathis, “An Architecture for Large-Scale Internet Measurement”, IEEE Communications Magazine p. 48-54, 1998

[24] A. Demers, S. Keshav, S. Shenker, "Analysis and simulation of a fair queueing algorithm", SIGCOMM Symposium on Communications Architectures and Protocols, ACM, pp. 1-12, September 1989

[25] H.Brandt, A. Eriksson, P. Evaldsson, G. Leijonhufvud, B. Ohlman, G. Olsson, J. Söderberg, E. Wedlund, “The SwitchLab Paper on Real-time IP Networking”, Switch Lab, Ericsson Radio Systems, April 1999

# A APPENDIX

## A.1 *Abbreviations*

| | |
|---|---|
| API | Application Program Interface |
| ASN.1 | Abstract Syntax Notation One |
| BE | Best Effort service |
| CAR | Committed Access Rate |
| CQ | Custom Queueing |
| CPU | Central Processing Unit |
| DiffServ | Differentiated Services |
| DNS | Domain Name System |
| DS | Differentiated Services |
| DSCP | DiffServ Code Point |
| FCS | Frame Check Sequence |
| FIFO | First In First Out |
| FTP | File Transfer Protocol |
| GPS | Global Positioning System |
| GSM | Global System for Mobile communication |
| GUI | Graphical User Interface |
| HTTP | Hyper Text Transfer Protocol |
| IETF | Internet Engineering Task Force |
| IntServ | Integrated Services |
| IOS | Cisco Internet Operating System |
| IP | Internet Protocol |
| MAC | Media Access Control |
| MIB | Management Information Base |
| moo | Measurable One-way Observer |
| MTU | Maximum Transfer Unit |
| PCM | Pulse Code Modulation |
| PHB | Per Hop Behavior |
| PQ | Priority Queueing |
| PRIO | Priority Queueing |
| PS | Premium Service |
| QoS | Quality of Service |

| | |
|---|---|
| RFC | Request For Comment |
| RTP | Real Time Protocol |
| SMI | Structure of Management Information |
| SNMP | Simple Network Management Protocol |
| Tcl | Tool Command Language |
| ToS | Type of Service |
| UDP | User Datagram Protocol |
| UTC | Coordinated Universal Time |
| WFQ | Weighted Fair Queueing |

## A.2 *A Scotty Script*

This section presents a semi-generic Scotty script which retrieve information from network elements via SNMP. The script supports both on demand and periodical collections and can easily be adapted to other MIBs and its MIB variables. Two types of variables are supported. Variables which hold instantaneous values and variables with accumulated values. For variables with accumulated values, the difference for each collection is automatically calculated. To adapt to other MIBs or variables, change the following code in the script:

```
# Load necessary management information base(s).

mib load CISCO-SMI.my
mib load CISCO-QUEUE-MIB.my

# Specify requested variables.

set variables {cQStatsDepth cQStatsMaxDepth}
set deltaVars {cQStatsDiscards}
```

All collected information is stored locally in a file with script name and date for easy access.

This particular script retrieves queue information from Cisco routers. It is started by the following command string:

```
./ciscoQueue.tcl -address 130.10.1.1 -community public -interval 1 -interfaces "1"
-queues "0 3" -days 1 -duration 2
```

The script will collect information one time from a router with IP address 130.10.1.1. Information from queue 0 and 3 at interface 1, will be retrieved every second during a 2 minute period.

The remaining part of this section outlines the whole code for this script.

```
#!/bin/sh
# the next line restarts using scotty \
exec scotty "$0" "$@"

# ciscoQueue.tcl --
#
# This file collects queue information from Cisco routers by means of
# SNMP request and store the information locally in a file.
#
# author: Mikael Torneus  Mikael.Torneus@era.ericsson.se
```

```
package require Tnm 2.1

# PrintStats --
#
# PrintStats is called whenever a SNMP get request is replied.
# The procedure calculates each requested variable and print
# the result to an output file.
#
# Arguments:
# job    The job object who issued the SNMP get request.
# err    Error status of the reply message.
# errInd Error index in the varbind list.
# vbl    Varbind list with variable and value.
#
# Results:
# Writes all variables in varbind list to an output file.

proc PrintStats {job err errInd vbl} {
   # Check if this job is finished, if so just return.

   if {[lsearch -exact [job info] $job] == -1} {
      return
   }

   # Fetch and calculate stored job data.

   set out [$job attribute -file]
   set rtime [expr [clock seconds] - [$job attribute -btime]]

   # write error message into file.

   if {$err != "noError"} {
      puts $out [format "# %u errorcode/index: %s/%d" $rtime $err $errInd]
      flush $out
      return
   }

   # Fetch each variable from vbl and store the new value in the file.
   # For deltaVars, just calculate the delta and store it.

   puts -nonewline $out [format "%u" $rtime]
   set ovbl [$job attribute -vbl]
   set deltaVars [$job attribute -deltaVars]
   foreach v $vbl ov $ovbl {
      set findDelta 0
      set val [lindex $v 2]
      set var [mib name [lindex $v 0]]
      set oval [lindex $ov 2]
      foreach dv $deltaVars {
         if {[string match $dv* $var] == 1} {
            set findDelta 1
         }
      }
      if {$findDelta} {
         puts -nonewline $out [format " %d" [expr $val - $oval]]
      } else {
         puts -nonewline $out [format " %d" $val]
      }
   }

   # Store the new vbl for the next SNMP response

   $job attribute -vbl $vbl
```

```
   puts $out ""

   return
}

# SnmpQueryProc --
#
# SnmpQueryProc issues an asynchronous SNMP get request using the
# attributes stored in the job object.
#
# Arguments:
# job    The job object who wants to issue a SNMP get request.
#
# Results:
# Issues an asynchronous SNMP get request.

proc SnmpQueryProc {job} {
   set session [$job attribute -session]
   set vbl [$job attribute -vbl]

   # Issue the request.

   $session get $vbl [subst {PrintStats $job %E %I {%V}}]

   return
}

# OpenStatFile --
#
# OpenStatFile creates a filename by concatenating the address, scriptName,
# interface and the current time. It also opens the file for writing.
#
# Arguments:
# address    The address of the peer.
# scriptName The name of the script.
# interface  Which interface in the peer
#
# Results:
# Creates and opens a file for storing of statistics.

proc OpenStatFile {address scriptName interface} {
   set clock [clock seconds]
   set fileName [format "%s-%s-%.2u-%s" $address $scriptName $interface \
      [clock format $clock -format "%Y.%m.%d"]]

   set code [catch {open $fileName w+} out]
   if $code {
      puts stderr [format "OpenStatFile: error %d: %s" $code $out]
      exit 1
   }

   # Flush after each new line.

   fconfigure $out -buffering line
   return $out
}

# CreateStatJobs --
#
# CreateStatJobs creates a job object for each interface.
# For each job object:
# - Create a vbl.
# - Open a new file to store all statistic data.
# - Collect base values and store it in the new file.
```

```
# CreateStatJobs waits for each job to finish and all SNMP requests
# to be completed before it returns.
# All open files are also closed before it returns.
#
# Arguments:
# session    Current SNMP session.
# iterations Number of samples.
# scriptName Name of the script currently running.
# interfaces A list of interfaces which is going to be monitored.
# duration   The duration in minutes of the measurement.
# interval   The time in ms. between each request.
# queues     Which queues are going to be monitored.
# variables  A list of variable names which the current value is of interest.
# deltaVars  A list of variable names which the delta is of interest.
#
# Results:
# Creates and starts a job for each interface. Each job writes its output
# to a file.

proc CreateStatJobs {session iterations scriptName interfaces duration \
   interval queues variables deltaVars} {

   # Create a job object for each interface.

   foreach interface $interfaces {
      # Prepare a vbl.

      set vbl {}
      foreach q $queues {
         foreach v $variables {
            lappend vbl $v.$interface.$q
         }
         foreach dv $deltaVars {
            lappend vbl $dv.$interface.$q
         }
      }

      # Open a new file.

      set file [OpenStatFile [$session cget -address] $scriptName $interface]
      set files($interface) $file

      # Calculates base values.

      set code [catch {$session get $vbl} result]
      if $code {
         puts stderr [format "CreateStatJob: error %d: %s" $code $result]
         exit 1
      }

      # Prepare a new file header.

      set clock [clock seconds]
      global tnm
      puts $file [format "# %s (based on scotty %s)" $scriptName \
         $tnm(version)]
      puts $file "#"
      set ipaddr [$session cget -address]
      if [catch {dns name $ipaddr} fqdn] {
         set fqdn $ipaddr
      }
      puts $file [format "# Agent:%s (%s)" $fqdn $ipaddr]
      puts $file [format "# Interface index:%s" $interface]
      puts $file [format "# Start:%s" [clock format $clock]]
```

```
      puts $file [format "# Duration:%s minute(s)" $duration]
      puts $file [format "# Interval:%s second(s)" \
         [expr $interval / 1000]]
      puts $file "#"
      puts $file "# Description of the columns in this file:"
      puts $file "#  column-index name base-value"
      puts $file [format "#%% 1 time %u" $clock]
      set i 2
      foreach v $result {
         puts $file [format "#%% %d %s %s" \
            $i [mib name [lindex $v 0]] [lindex $v 2]]
         incr i
      }
      puts $file "#"

      # Create the job object.

      set job [job create]
      $job attribute -file $file
      $job attribute -session $session
      $job attribute -scriptName $scriptName
      $job attribute -interface $interface
      $job attribute -vbl $result
      $job attribute -deltaVars $deltaVars
      $job attribute -btime $clock

      # Measurements show that the interval is 10 ms. longer than specified.
      # The interval is thus reduced by 10 ms.

      $job configure -command [subst {SnmpQueryProc $job}]\
         -interval [expr $interval - 10] -iterations $iterations
   }

   # Wait for the jobs to finish and also make sure that all
   # asynchronous SNMP requests are completed.

   job wait
   $session wait

   # Close all open files.

   foreach interface $interfaces {
      set file $files($interface)
      if {$file != ""} {
         close $file
      }
   }
   puts stderr "Measurements are finished."

   return
}

# Sleep --
#
# Sleep sleeps for sec seconds before it returns.
#
# Arguments:
# sec    Number of seconds to sleep.
#
# Results:
# Sleep for sec seconds.

proc Sleep {sec} {
   after [expr $sec * 1000] {set done 1}
```

```
   vwait done
}

# Main Program --
#
# The main program does the following:
# - Checks the command line arguments.
# - Loads the necessary mibs.
# - Creates the snmp session.
# - Enters the daily loop.
#
# Arguments:
# -h                    Shows this help text.
# -address    address   The address of the SNMP peer.
#                       It could be an IP-address or a hostname.
#                       Default is localhost.
# -community  community This identifies the sender of the SNMP message.
#                       Default is public.
# -duration   duration  The duration of the measurement in minutes.
#                       Default is 2 minutes.
# -interval   interval  The interval in seconds between each sample.
#                       Default is 30 seconds.
# -interfaces interfaces The interfaces who is going to be monitored.
#                       Must be specified (e.g. "1 2 3").
# -days       days      Number of days the measurement will run.
#                       If 0 days is specified, the measurement will run
#                       continously (e.g. every day). Default is 1 days.
# -queues     queues    Which queues will be monitored.
#                       Must be specified (e.g. "0 3").
# -date       date      The start date of the measurement.
#                       The start date is specified as mm/dd/yyyy
#                       (e.g. 02/15/2000). Default is today's date.
# -time       time      The start time of the measurement.
#                       The start time is specified as hh:mm:ss
#                       (e.g. 14:35:00). Default is "now".
#
# Note. All default values can be changed below.
#
# Results:
# Creates an output file for each interface where statistic data are stored.

# Set default values.

set address localhost
set community public
set interval 30000
set days 1
set interfaces {}
set queues {}
set duration 2
set clock [clock seconds]
set currentDate [clock format $clock -format "%m/%d/%Y"]
set date $currentDate
set time now

# If no options or -h is specified.

if {($argc <= 0) || ([string match -h $argv] == 1)} {
   puts stderr "Options: "
   puts stderr [format "-h Shows this help text."]
   puts stderr [format "-address address The address of the SNMP peer."]
   puts stderr [format "It could be an IP-address or a hostname."]
   puts stderr [format "Default is %s." $address]
   puts stderr [format "-community community This identifies the sender of\
```

```
      the SNMP message."]
   puts stderr [format "Default is %s." $community]
   puts stderr [format "-duration duration The duration of the measurement\
      in minutes."]
   puts stderr [format "Default is %s minutes." $duration]
   puts stderr [format "-interval interval The interval in seconds between\
      each sample."]
   puts stderr [format "Default is %s seconds." [expr $interval/1000]]
   puts stderr [format "-interfaces interfaces The interfaces who is going\
      to be monitored."]
   puts stderr [format "Must be specified (e.g. '1 2 3')."]
   puts stderr [format "-days days Number of days the measurement will\
      run."]
   puts stderr [format "If 0 days is specified, the measurement will\
      run"]
   puts stderr [format "continously (e.g. every day). Default is %s\
      days." $days]
   puts stderr [format "-queues queues Which queues will be monitored."]
   puts stderr [format "Must be specified (e.g. '0 3')."]
   puts stderr [format "-date date The start date of the measurement."]
   puts stderr [format "The start date is specified as mm/dd/yyyy"]
   puts stderr [format "(e.g. 02/15/2000). Default is today's date."]
   puts stderr [format "-time time The start time of the measurement."]
   puts stderr [format "The start time is specified as hh:mm:ss"]
   puts stderr [format "(e.g. 14:35:00). Default is 'now'."]
   exit 1
}

# Check the command line arguments.

for {set i 0} {$i < $argc} {incr i} {
   switch -- [lindex $argv $i] {
      -address {
         incr i
         set address [lindex $argv $i]
      }
      -community {#
         incr i
         set community [lindex $argv $i]
      }
      -interval {
         incr i
         set interval [expr 1000 * [expr int([lindex $argv $i])]]
      }
      -interfaces {
         incr i
         set interfaces [lindex $argv $i]
      }
      -days {
         incr i
         set days [lindex $argv $i]
      }
      -queues {
         incr i
         set queues [lindex $argv $i]
      }
      -duration {
         incr i
         set duration [expr int([lindex $argv $i])]
      }
      -date {
         incr i
         set date [lindex $argv $i]
      }
```

```
      -time {
         incr i
         set time [lindex $argv $i]
      }
      default {
         puts stderr [format "invalid option(%d): %s" $i [lindex $argv $i]]
         exit 1
      }
   }
}
if {[llength $interfaces] < 1} {
   puts stderr "No interfaces are specified!"
   exit 1
}
if {[llength $queues] < 1} {
   puts stderr "No queues are specified!"
   exit 1
}
if {$interval <= 0} {
   puts stderr "Interval must be greater than zero !"
   exit 1
}
if {$duration <= 0} {
   puts stderr "Duration must be greater than zero !"
   exit 1
}

# Check if measurement is scheduled for today.

if {[clock scan $date] == [clock scan $currentDate]} {
   set scheduledTime [clock scan $time]
} else {
   set scheduledTime [expr [clock scan $date] +\
      [expr [clock scan $time] - [clock scan $currentDate]]]
}
if {$scheduledTime < $clock} {
   puts stderr "Scheduled start time has already past"
   exit 1
}

# Show the specified options.

puts stderr "Options: "
puts stderr [format "-address: %s" $address]
puts stderr [format "-community: %s" $community]
puts stderr [format "-duration: %s minute(s)" $duration]
puts stderr [format "-interval: %s second(s)" [expr $interval / 1000]]
puts stderr [format "-interfaces: %s" $interfaces]
puts stderr [format "-days: %s" $days]
puts stderr [format "-queues: %s" $queues]
puts stderr [format "-date: %s" $date]
puts stderr [format "-time: %s" $time]

# Load necessary management information base(s).

mib load CISCO-SMI.my
mib load CISCO-QUEUE-MIB.my

# Specify requested variables.

set variables {cQStatsDepth cQStatsMaxDepth}
set deltaVars {cQStatsDiscards}

# Find out the name of the script.
```

```
# This is used as part of the output file.

if {[string match ./* $argv0] == 1} {
   set scriptName [string range $argv0 2 [string length $argv0]]
} else {
   set scriptName $argv0
}

# Calculate number of iterations for the measurement duration.
# Add two extra so you don't lose the startvalue and the last collection.

set iterations [expr (($duration * 60) / ($interval / 1000)) + 2]

# Create a session for the peer.

set code [catch \
   {snmp session \
   -address $address -community $community \
   -delay 0 -window 4} session]
if $code {
   puts stderr [format "SNMP session: %s" $session]
   exit 1
}

# Wait for start time of the measurement.

set secs [expr $scheduledTime - $clock]
if {$secs > 0} {
   Sleep $secs
}

# Restart the statistic collection jobs once a day.

set i 0
while {($days == 0) || ($i < $days)} {
   set code [catch {CreateStatJobs $session $iterations $scriptName\
      $interfaces $duration $interval $queues $variables $deltaVars} msg]
   if $code {
      puts stderr $msg
      exit 1
   }
   incr i
   if {$days == 1} {
      exit 0
   }
   if {($days != 0) && ($i >= $days)} {
      exit 0
   }
   set secs [expr 86400 - $clock % 86400]
   Sleep $secs
}
exit 0
```

## A.3 *Router Configuration*

This section describes in a more detailed way how the router was configured. Below shows a printout of the used router configuration.

Premium Service for Priority Queueing and Custom Queueing was achieved by priority lists. Ethernet interface 1/0 has a priority-group 1 connected. The priority-group or priority-list attachs access-list 100 - 107. These access-lists only permit traffic with a certain prece-

dence bit set, e.g. access-list 101 only permits traffic with precedence 1 (priority). Ethernet interface 0/0 has a custom-queue-list 1 connected which attachs the access-lists in the same way as Priority Queueing.

Premium Service for Weighted Fair Queueing was achieved by enabling fair-queue for Ethernet interface 0/1. No access-lists are needed, the WFQ algorithm priorize the weight for each flow according to the IP Precedence bits automatically.

The moo software could not use different precedence bit settings concurrently and thus was solved by the router software. Access-lists 115 and 117 catch traffic to UDP port 10005 and 10007 respectively. These two access-lists are used by CAR to change the precedence bits for incoming traffic at interface Ethernet0/1, i.e. traffic for UDP port 10005 and 10007 are change to precedence value 5 (critical) and 7 (network), respectively. The probe traffic must use either port number to recieve a correct treatment from the network. An arbitrary UDP port number for each traffic stream can be specified in the moo software.

Alarms by means of *traps* were enabled by snmp-server and rmon commands.

Here follows a printout of the current router configuration.

```
cisco1#show running
Building configuration...

Current configuration:
!
version 12.0
service config
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname cisco1
!
enable secret 5 $1$8Ha3$Q5obPZ6W5T5Pq2JHZ9lNO0
enable password diffserv
!
!
!
!
!
ip subnet-zero
ip domain-name lab.net
ip name-server 130.10.1.11
!
cns event-service server
!
!
!
process-max-time 200
!
interface Ethernet0/0
ip address 130.10.4.1 255.255.255.0
ip directed-broadcast
no ip mroute-cache
custom-queue-list 1
no cdp enable
!
interface Ethernet0/1
ip address 130.10.1.1 255.255.255.0
```

```
ip directed-broadcast
rate-limit input access-group 115 10000000 10000 10000 conform-action set-prec-
transmit 5 exceed-action set-prec-transmit 5
rate-limit input access-group 117 10000000 10000 10000 conform-action set-prec-
transmit 7 exceed-action set-prec-transmit 7
no ip mroute-cache
fair-queue 64 16 0
no cdp enable
!
interface Ethernet1/0
ip address 130.10.3.1 255.255.255.0
ip directed-broadcast
no ip mroute-cache
priority-group 1
no cdp enable
!
router rip
network 130.10.0.0
!
ip classless
no ip http server
!
access-list 100 permit ip any any precedence routine
access-list 101 permit ip any any precedence priority
access-list 102 permit ip any any precedence immediate
access-list 103 permit ip any any precedence flash
access-list 104 permit ip any any precedence flash-override
access-list 105 permit ip any any precedence critical
access-list 106 permit ip any any precedence internet
access-list 107 permit ip any any precedence network
access-list 115 permit udp any any eq 10005
access-list 117 permit udp any any eq 10007
queue-list 1 protocol ip 1 list 107
queue-list 1 protocol ip 16 list 100
queue-list 1 default 16
queue-list 1 queue 16 limit 80
priority-list 1 protocol ip high udp snmp
priority-list 1 protocol ip low list 100
priority-list 1 protocol ip low list 101
priority-list 1 protocol ip normal list 102
priority-list 1 protocol ip normal list 103
priority-list 1 protocol ip medium list 104
priority-list 1 protocol ip medium list 105
priority-list 1 protocol ip high list 106
priority-list 1 protocol ip high list 107
!
snmp-server engineID local 0000000902000003094286EE0
snmp-server community public RO
snmp-server enable traps snmp
snmp-server host 130.10.1.14 public
snmp-server manager
rmon event 1 trap public owner config
rmon alarm 1 ifEntry.11.3 1 delta rising-threshold 200 1 falling-threshold 200 1
owner config
!
line con 0
exec-timeout 0 0
transport input none
line aux 0
line vty 0 4
password diffserv
login
!
!
```

```
no scheduler allocate
end
```