A Study of Quality of Service dynamics in the Internet

The effects of Routing & Link State Update Policies

Vahid Niazian

Supervisor: Prof. dr. ir. P. Van Mieghem
Advisors: ir. F. A. Kuipers
          ir. B. Lekovic

July, 2001
M. Sc Thesis No: PVM 2001 – 009
Preface

This MSc thesis has been accomplished at the Delft University of Technology, faculty of Information Technology and Systems, Workgroup Telecommunications and Traffic-Control Systems (TVS).

The focus of this thesis is studying the network performance in terms of delivering QoS with regard to different routing algorithms and link state update policies.

The conclusions of this thesis are based on the simulation results, obtained from the Quality of Service routing simulator version 1.1 (QRS 1.1), within an Integrated Services architecture.

Acknowledgements

With these last words, which I am writing in my thesis, I would like to thank Prof P. Van Mieghem, Fernando Kuipers and Bojan Lekovic. The time and the support they gave me, not only resulted in this thesis, but also in a friendship that will continue even though this thesis has come to its end.

And my thanks also goes to my family who supported me in every possible way and to all my friends, specially ‘my brother’ Nader Gaffari for his unconditional support and I apologize for not having enough time for them in the past few months.
## Abbreviations and acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ack packet</td>
<td>Acknowledgement packet</td>
</tr>
<tr>
<td>Adspeck</td>
<td>Advertise Specification</td>
</tr>
<tr>
<td>AF-PHB</td>
<td>Assured Forwarding Per Hop Behaviour</td>
</tr>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode</td>
</tr>
<tr>
<td>BE</td>
<td>Best Effort</td>
</tr>
<tr>
<td>CBQ</td>
<td>Class Based Queuing</td>
</tr>
<tr>
<td>CL</td>
<td>Controlled Load</td>
</tr>
<tr>
<td>CS-PHB</td>
<td>Class Selector Per Hop Behaviour</td>
</tr>
<tr>
<td>DS</td>
<td>Differentiated Services</td>
</tr>
<tr>
<td>DSCP</td>
<td>Differentiated Services Code Point</td>
</tr>
<tr>
<td>ECB</td>
<td>Equal Class Based</td>
</tr>
<tr>
<td>EF-PHB</td>
<td>Expedited Forwarding Per Hop Behaviour</td>
</tr>
<tr>
<td>Error spec</td>
<td>Error Specification</td>
</tr>
<tr>
<td>EWMA</td>
<td>Exponential Weighted Moving Average</td>
</tr>
<tr>
<td>FEC</td>
<td>Forwarding Equivalent Class</td>
</tr>
<tr>
<td>FF</td>
<td>Fixed Filter Style</td>
</tr>
<tr>
<td>Filter spec</td>
<td>Filter Specification</td>
</tr>
<tr>
<td>Flowspec</td>
<td>Flow Specification</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GS</td>
<td>Guaranteed Service</td>
</tr>
<tr>
<td>HT</td>
<td>Hold Timer</td>
</tr>
<tr>
<td>IntServ</td>
<td>Integrated Services</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPv4</td>
<td>Internet Protocol version 4</td>
</tr>
<tr>
<td>IPv6</td>
<td>Internet Protocol version 6</td>
</tr>
<tr>
<td>LC</td>
<td>Lowest Cost routing algorithm</td>
</tr>
<tr>
<td>LDP</td>
<td>Label Distribution Protocol</td>
</tr>
<tr>
<td>LSU</td>
<td>Link State Update</td>
</tr>
<tr>
<td>LSUp</td>
<td>Link State Update Policy</td>
</tr>
<tr>
<td>MARS</td>
<td>MAryland Routing Simulator</td>
</tr>
<tr>
<td>MPLS</td>
<td>Multi Protocol Label Switching</td>
</tr>
<tr>
<td>OPWA</td>
<td>One Pass With Advertising</td>
</tr>
<tr>
<td>PB</td>
<td>Periodic Based</td>
</tr>
<tr>
<td>PHB</td>
<td>Per Hop Behaviour</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>QOSPF</td>
<td>QoS Open Shortest Path First</td>
</tr>
<tr>
<td>QRS 1.1</td>
<td>Quality of Service Routing Simulator, version 1.1</td>
</tr>
<tr>
<td>Resv Error Packet</td>
<td>Reservation Error Packet</td>
</tr>
<tr>
<td>Resv Packet</td>
<td>Reservation Packet</td>
</tr>
<tr>
<td>Resv Tear Packet</td>
<td>Reservation Tear Packet</td>
</tr>
<tr>
<td>Rspec</td>
<td>Request Specification</td>
</tr>
<tr>
<td>RSVP</td>
<td>Resource ReserVation protocol</td>
</tr>
<tr>
<td>Sacmra ER</td>
<td>Samcra Explicit Routing</td>
</tr>
<tr>
<td>Samcra hbh</td>
<td>Samcra Hop-By-Hop</td>
</tr>
<tr>
<td>Samcra</td>
<td>Self-Adaptive Multiple Constraints Routing Algorithm</td>
</tr>
<tr>
<td>SE</td>
<td>Shared Explicit Style</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>TB</td>
<td>Threshold Based</td>
</tr>
<tr>
<td>TCA</td>
<td>Traffic Conditioning Agreement</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TOS</td>
<td>Type Of Service</td>
</tr>
<tr>
<td>Tspec</td>
<td>Traffic Specification</td>
</tr>
<tr>
<td>TTL</td>
<td>Time To Live</td>
</tr>
<tr>
<td>UCB</td>
<td>Unequal Class Based</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>WF</td>
<td>Wildcard Filter</td>
</tr>
<tr>
<td>WFQ</td>
<td>Weighted Fair Queuing</td>
</tr>
<tr>
<td>xDSL</td>
<td>A general name for all variations of ‘Digital Subscriber Loop’</td>
</tr>
</tbody>
</table>
Table of contents

1. Introduction .................................................. 1
2. A QoS architecture .......................................... 3
3. Integrated Services ........................................... 5
   3.1 The general description .................................. 5
      3.1.1 Request for reservation .......................... 5
      3.1.2 Reservation .................................... 7
      3.1.3 Maintaining the reservation .................... 8
      3.1.4 Terminating a reservation ..................... 8
      3.2 Components ...................................... 10
         3.2.1 RSVP ....................................... 10
         3.2.2 Path packet .................................. 12
         3.2.3 Path Error packet ............................ 13
         3.2.4 Resv Packet .................................. 13
         3.2.5 Resv Error .................................... 14
         3.2.6 Packet scheduler ................................
            3.2.6.1 Class based queuing ..................... 14
            3.2.6.2 Weighted fair queuing .................. 15
      3.2.7 Routing ...................................... 15
         3.2.7.1 Routing through RSVP-unaware nodes .... 15
      3.2.8 Aggregation of flows ........................... 16
      3.2.9 Confirmation .................................. 17
      3.3 Some weaknesses of IntServ ......................... 18
4. Differentiated Services ...................................... 19
   4.1 Service Level Agreement ............................... 20
   4.2 Packet classifier and traffic conditioner ............ 21
      4.2.1 Packet classifier ............................... 21
      4.2.2 Meter ......................................... 22
      4.2.3 Marker ........................................ 24
   4.3 The location of traffic conditioners and classifiers .. 24
      4.3.1 At the boundary nodes of a DS domain ........ 25
      4.3.2 At the interior nodes of a DS domain .......... 25
      4.3.3 At the source ................................... 25
      4.3.4 At non-DS-capable nodes ....................... 25
   4.4 Per hop behaviour .................................... 25
      4.4.1 Expedited Forwarding (EF) ..................... 26
      4.4.2 Assured Forwarding group (AF) ................ 26
      4.4.3 The Class Selector PHB Group .................. 26
5. Multi Protocol Label Switching ................................ 28
   5.1 Label assignment and distribution ...................... 28
   5.2 Label Distribution Protocols .......................... 29
   5.3 Routing ............................................ 29
      5.3.1 Hop by hop ..................................... 29
      5.3.2 Explicit routing ................................ 29
   5.4 Invalid labels ....................................... 30
   5.5 MPLS in a Differentiated Services domain ........... 30
6. QoS Routing in IntServ ...................................... 31
   6.1 QoS routing algorithm, Samcra ........................ 31
   6.2 QoS Routing Simulator ................................ 31
      6.2.1 Routing ......................................... 31
      6.2.2 Link state update policy ....................... 32
7. Simulation results ........................................... 35
   7.1 Simulation plan ....................................... 35
Part I

1. A general view

2. Network configuration
   2.1 A general overview of the input / output file
   2.2 A more detailed view of the components
      2.2.1 Node
      2.2.2 LINK
      2.2.3 Resource Management, RM
      2.2.4 Resource ReserVation Protocol, RSVP
      2.2.5 Quality of service OSPF, QOSPF
      2.2.6 Real-time Source
      2.2.7 Real-time Sink
      2.2.8 Link Cost Component
      2.2.9 Performance Monitor

Part II

3. The structure of the program regarding Routing and Link state update policy
   3.1 A general overview of QRS1.1 regarding real-time traffic
   3.1.2 Network representation
      3.1.2.1 Global Topology Table of cost
      3.1.2.2 Flow Global Topology Table
      3.1.2.3 Local Topology Table
      3.1.2.4 Flow Routing Table
      3.1.2.5 Flow Table
   3.2 Routing in QRS1.1
      3.2.1 Lowest cost routing algorithm
      3.2.2 Widest bandwidth routing algorithm
      3.2.3 Samcra hop-by-hop
         3.2.3.1 gtt matrix (global topology table)
         3.2.3.2 Adding a new c-file
         3.2.3.3 Adjacency list
         3.2.3.4 Source

Appendix. Additional Manual of QRS

Introduction

Part I

1. A general view

2. Network configuration

Part II

3. The structure of the program regarding Routing and Link state update policy
3.2.3.5 Destination ............................................. 81
3.2.3.6 Adjacency lists ........................................ 81
3.2.3.7 Number of metrics .................................... 82
3.2.3.8 Constraints ............................................. 82
3.2.4 Samcra source-based .................................... 83
   3.2.4.1 Difference of path between Samcra hbh and sb 84
3.2.5 Additional codes to record the paths and costs ....... 85

Part III ..................................................................... 86
4. Link state update .................................................. 86
1. Introduction

From its start, the Internet has grown considerably in size. The service available on the current Internet is of a best effort nature, where no guarantees are given to the users for the sake of fairness and simplicity. But after its early growth, the Internet is now passing a threshold in its evolution to become a more reliable and mature network, which not only supports the current best effort services but also in a large extent, more and more advanced services.

A first step in this evolution is taken by implementing wider bandwidth connections, which are made possible with the new technologies (e.g., xDSL technologies and fiber). These technologies, which are in the early ages of their evolution, have a very important contribution in making both the content providers and the users more data-aware. There are still two major problems that prevent these technologies from taking the Internet further in its evolution:

1. It is extremely costly, in terms of money and time, to provide every Internet user with wide bandwidth solutions. Although many believe, including the author of this report, that this eventually will happen, providing larger bandwidth is not a long term solution, which brings us to the second problem:

2. It is seen in the past that the larger the bandwidth gets, the more bandwidth consuming services will be deployed, which again will result in a shortage of available bandwidth. The moment that this problem is being solved by providing even larger bandwidths, an endless loop is formed.

However, it must be noted that, whatever the long-term solution will be, providing higher bandwidth is an essential part of that solution.

The second type of solution that is now being studied, but not yet implemented, is changing the structure of the current Internet. With structure is meant the way different users or their generated packets are being handled by the network. This includes a change in the applications using the Internet as well as different hard or software (e.g. hard and software of the routers) in the network that handle the packets. Having in mind that Internet has become a worldwide network, the introduced solutions should make as few changes as possible and at the same time be relatively simple to implement in the current Internet architecture.

These new solutions will allow the users to choose the kind of service they require, which in turn gives the service providers the opportunity to have differentiated billing systems for different customers. Setting up a Virtual Private Network (VPN) could also be made possible, where the companies do not have to implement a whole private network but use the Internet instead as their VPN.

The goal of this thesis is to study routing in one of the QoS architecture, called ‘Integrated Services’. Integrated Services uses a hop-by-hop routing algorithm, which is not Quality of Service-aware. With Quality of Service (QoS) in this
thesis is meant the ability to deliver certain services over the Internet, based on the agreement that is made between the user and the network administrator. The user can be a person, a company or another Internet Service Provider.

This thesis studies the use of a QoS-aware routing algorithm, called Samcra [1] in Integrated Services architecture. This is done on a hop-by-hop as well as an explicit routing basis. With explicit routing we mean that the source calculates the path and that the intermediate routers use this pre-calculated path to forward the packets to the next hop.

Further we study the effect of link state update policies on the performance of the network. This is done with different routing algorithms.

Simulation is the main tool for studying QoS in this thesis. The name of the simulator used is ‘QoS Routing Simulator’ (QRS version 1.1).

The thesis is divided in 8 chapters. Chapter 2 describes the general requirements of a network to deliver QoS.

Integrated Services will be discussed in chapter 3, where also some disadvantages of this architecture will be discussed.

Solving the problems in Integrated Services has led to a new architecture, called ‘Differentiated Services’, which will be discussed in chapter 4. Chapter 5 gives a brief overview of a traffic management solution, called Multi Protocol Label Switching (MPLS) and also refers to a possible integration between MPLS and Differentiated Services. An introduction to Samcra and the used simulator is given in chapter 6. Chapter 7 presents the simulation plan, after which the simulation results are presented and discussed. The conclusions about the results and the simulator are treated in chapter 8 and finally chapter 9 gives recommendations for future work for Master of Science students.
2. A QoS architecture

Much research has been and is being done over the last few years to find a solution to satisfy the needs of (future) customers and applications on the Internet. Although there are fundamental differences between the proposed architectures, many believe that such an architecture must contain several common components in order to deliver the QoS needed. This chapter will describe five components [2], which will be discussed further related to two different architectures in chapters 3 and 4.

Flow specification

An application requiring QoS should be able to inform the network about the type of flow it wants to generate and its QoS requirements. The network should in turn be able to understand this request. Therefore, there must be a common language between the source and the network. The requests of a source regarding its requirements will be specified in its Flow specification (flowspec). There are different proposals for such a flowspec.

Routing

In every communication session there are one or many sources sending data to one or more receivers. As these sources and receivers almost never have a direct connection, routing must be performed on the packets by the network to find the path from the source to the destination both in case of unicast and multicast. There are routing algorithms that are QoS-aware and algorithms that are not.

Resource reservation

To deliver the packets the service they require, the network should be able to reserve some resources (e.g., buffer space at the routers) along the path to be able to give certain guarantees regarding the handling of the packets.

Admission control

When a source informs the network about its requirements by sending a flowspec, before the network can reserve resources, it should check whether it has enough resources to fulfill the requirements of the source. Admission control must also check whether accepting a new request will effect its other reservations, in which case the request will be rejected.

Packet scheduling

When a source is allowed to send, scheduling techniques make sure that the packets receive the service, which they requested (or the customer is paying for). After the transmission of every packet, the packet scheduler decides which packet to transmit next. This decision is based on the type of service the packets require. In this way some packets get a priority treatment above other packets or certain
packets are dropped or degraded in service, because of crossing a certain limit of one or more parameters (e.g., bytes/s).

In the next two sections, these five components will be discussed in the context of two architectures, Integrated Services and Differentiated Services. As mentioned in the introduction, the main objective of this thesis will be routing in the Integrated Services architecture.
3. Integrated Services

3.1 The general description

Integrated services (IntServ) [3] [4] [5] is a receiver driven Internet architecture, where the source informs the destination about its intention to communicate and the way it wants to communicate, but it is the destination that must accept this request and if necessary change this request to make it suitable for itself and eventually reserve resources along the path for the desired communication. This reservation of resources along a path should be done by the routers, which also have to maintain the state of every flow being handled by them. If there are many flows present, as could be the case in backbone networks, maintaining state information on every flow becomes unfeasible.

Along a communication path, there are many components and controls that are interacting with each other and because of their complex interaction, it is difficult to explain each of them separately. This is the reason why I have chosen to explain Integrated Services by following a communication session from its start till its termination and in this way explain the components along the path of the flow. An immediate explanation of each of these components, when they are mentioned for the first time, will introduce many sub-chapters, which makes it difficult to form a total understanding of Integrated Services. So from paragraph 3.1.1, the actions during a session are explained and each time a component is mentioned, there will be a reference to a specific paragraph in paragraph 3.2, where additional information about that component can be found. Finally paragraph 3.3 discusses several disadvantages of IntServ.

3.1.1 Request for reservation

When a source, or actually an application, wants to set up a communication with a certain destination, the first step is informing the destination of its intention. The source can choose between three services available within IntServ:

1- Guaranteed Service (GS)
With guaranteed service the user is given certain guarantees about the maximum delay or minimum bandwidth through the network. This means that the packets receiving guaranteed service will not be discarded even during network congestion, assuming that the packets do not exceed their specified traffic parameters. Applications with video streaming may use this service. Guaranteed service needs both the Tspec and the Rspec to be defined (see paragraph 3.2.1).

2- Controlled-Load (CL)
In this case, no hard guarantees are given to the user but all effort is made to give the user the same amount of resources, as he would get in a situation where the network is not heavily loaded. The applications that can use this service are like the current Internet applications. The controlled-load service only tries to deliver the amount of bandwidth on which an agreement has been established between the user and the network operator. Packets receiving this service will get a priority
treatment above the best-effort packets. Controlled-load service needs only a Tspec definition (see paragraph 3.2.1).

3- Best-effort (BE)
As the third service that a user can apply for in an IntServ environment, best-effort service can be mentioned. This service is currently delivered on the Internet. This service may be used for receiving emails. It should be prevented that the best-effort packets get an unfair treatment, because of the packets belonging to a higher service class.

The signalling protocol that is now being used within IntServ to reserve resources along a path is called Resource ReserVation Setup Protocol, RSVP (paragraph 3.2.1). In order to use the services available within IntServ, the source must be RSVP-aware to be able to translate its needs (required bandwidth, maximum delay, etc.) and inform the network as well as the destination about its requirements in a way that all the components understand. So it is also necessary that the routers along the path as well as the destination itself are RSVP-aware. This awareness is only needed in case that guaranteed or controlled-load service are requested.

It is important to note that RSVP is just a signalling protocol used within IntServ for reserving resources along a path and that it can be replaced by some other capable reservation protocol.

When a RSVP-aware source wants to communicate with a certain destination, it will introduce a so-called Path packet (paragraph 3.2.2) and send it to the network to ask the network and the receiver for permission to set up a communication session. Such a Path packet carries information about the requirements of the flow\(^1\) that the source wants to generate. These requirements can be the minimum bandwidth or the maximum delay. This Path packet is received by the first router, where it is given to the Admission control and Policy control. Policy control checks whether the source has the administrative right to use that certain network or that certain service. Admission control checks the requirements in the Path packet and determines if there are enough resources in order to be able to handle that flow.

If the result of one of these checks is negative, a Path Error packet (paragraph 3.2.3) is sent back to the source. The source knows now that its request has been rejected. The source can now:
- Forget about setting up a communication session with that certain destination;
- Wait for a certain time and try again;
- Adjust its requirements and try to get a new permission with a new Path packet.

If both the results of the policy control and the admission control are positive, the router will perform the following actions:

---

\(^1\) By ‘flow’ in this thesis is meant a stream of IP packets between a source and a destination (port).
• The router makes a temporary reservation of its resources for that flow;
• The timer of that path is reset. The expiration of this time triggers the deletion of the path. Therefore the Path message has to be sent periodically as long as the path is alive, in order to refresh the path state.
• If the Path packet came from a previous router, the current router keeps the address of this previous router;
• If two or more Path packets have arrived at the router, the router can, if possible, merge (paragraph 3.2.8) the requirements of these two flows and make a new Path packet to send to the next router;
• According to its routing table, the router will find a next hop for the Path packet based on the destination address of the packet (paragraph 3.2.7). The packet is then sent to the next hop.

When this (new) Path packet has reached the next router, the same process will take place again at that router. In this way, if there are enough resources available in the network, a Path packet can eventually reach the destination.

3.1.2 Reservation

Once the destination has received the Path packet, it is informed about the intention of the source and its flow properties. The destination can now decide whether it wants a communication session with that source. It also checks if the flow properties of the source are suitable for him and if not adjusts these properties.

Once the destination has decided to communicate, it sends back a packet called Reservation packet (Resv packet, paragraph 3.2.4) back to the source. This Resv packet now contains the requirements of the destination with respect to that flow. One of these properties could be the minimum bandwidth required, which is not necessarily the same as the required bandwidth advertised in the Path packet!

In this Resv packet, the destination can also ask the network to inform him if his request is accepted by the network. This is called a confirmation (paragraph 3.2.9). However, this confirmation forms no guarantee that the flow can indeed be handled by the network.

Once a Resv packet reaches the first router, again the packet is checked by the admission control and the policy control. If the result of one of these checks is negative, a Resv Error packet (paragraph 3.2.5) is sent back to the destination. If both the results are positive, the router performs the following actions:
• It makes a ‘hard’ reservation of its resources for that flow;
• If the router has received several Resv packets at a certain time, it can merge two or more of these requests, if possible and make a new Resv packet and send it to the next hop. This is of course done, if possible (paragraph 3.2.8);
• The router will not look in its routing table to route this packet. This Resv packet must follow the reverse path of the Path packet. As explained above, the routers have already kept the address of the router that sent them a Path packet. So the Resv packet will automatically be sent to this next router.
When the Resv Packet reaches the next router, the same process will take place. In this way, if there are enough resources along the path, the Resv packet will reach the source.

When a Resv packet has reached the source it means that along the path every router is informed about the requirements of the flow that is going to be generated and has reserved enough resources. The source can now start sending the flow packets to the destination.

3.1.3 Maintaining the reservation

In order to keep a path alive, the source and the destination regularly send respectively Path and Resv packets along the path to refresh the path state. In this way the resources stay reserved as long as necessary.

When packets arrive at a router, the Packet classifier checks these incoming packets according to the information received by the filter specification. Packet classifier determines then, which packets belong to which flow. After sorting these incoming packets, they are passed to Packet scheduler (paragraph 3.2.6). The packet scheduler is concerned with how the incoming packets should be delivered to the output links of the router according to their required service in terms of packet priority. This scheduling is done according to some scheduling algorithm (paragraph 3.2.6.1, 3.2.6.2).

3.1.4 Terminating a reservation

When a flow path has been set up with all its resource reservations, the reserved path should not exist forever and must be terminated when it is no longer needed. There are several ways in which a path can be terminated.

1- When a source decides to break up the communication, it can send a Path Tear packet (paragraph 3.2.3) down to the destination. Every router along the path that receives this packet, will release the resources reserved for that certain flow. When a Path Tear packet has reached the destination, it means that the whole path has been released.

2- When a destination wants to terminate a communication session, it sends a Resv Tear packet to the source. With this packet every intermediate router will release its resources reserved for that flow and forwards the packet to the next router. When the Resv Tear packet has reached the source, it means that the whole path has been released.

3- If a router does not receive a refresh packet before the end of Time To Live (TTL) of the path, the path is terminated by the router itself. The router will then send a Path Tear packet to the receiver and a Resv Tear packet to the source.
One of the robustness properties of RSVP is that if a Tear packet is lost in the network, the path will be terminated anyway. When a path must be terminated and one of the routers does not receive a Tear packet, the path will eventually be terminated once the TTL of that path in that router is expired.
3.2 Components

3.2.1 RSVP

RSVP is the signalling protocol used with IntServ [6]. It defines certain packets and rules to deliver the desired QoS. A RSVP-aware source can advertise its required service to the network. RSVP supports both unicast as well as many-to-many multicast applications and has the ability to adapt dynamically to the changes of the users in a multicast session.

A basic RSVP packet contains a Flow specification and a Filter specification. Flow specification defines the service required by the sender and the filter specification sets the parameters in the node (router) in order to select the right packets between all the incoming packets at that node.

The flow specification defines the kind of service required by the source or the destination and some parameters that can be set for that service.

Five of the parameters that can be set by the flow specification are:

Traffic specification (Tspec):
Tspec contains information about the data flow that will be generated. Tspec is used by the node to check if the flow can be accepted or not. Tspec contains the following parameters:

\[ R: \]
Data rate of the flow measured in bytes of IP datagrams per second.

\[ B: \]
Bucket size measured in bytes. This indicates that at time \( t \) the amount of data sent by the source is smaller than \( (R \cdot t + B) \) for all \( t \)'s;

Peak traffic rate \( p \)

Minimum policed unit \( m \):
All packets smaller than size \( m \) are treated as being of size \( m \) for purposes of resource allocation and policing. The purpose of this parameter is to allow reasonable estimation of the per-packet resources needed to process a flow. \( m \) is measured in bytes.

Maximum packet size \( M \):
The maximum packet size, \( M \), is the biggest packet that will conform to the traffic specification. Any packets of larger size sent into the network may not receive the service required, since they are considered to be not conform the traffic specification. \( M \) is also measured in bytes.

Reserve specification (Rspec):
Defines the required QoS in terms of
R: data rate of the flow  
S: maximum delay of the flow (Slack delay)

3.2.1.1 A more detailed view of a RSVP message

A RSVP message consists of two parts: a Header and an Object. Several contents of these two parts will be discussed here.

**IP Header:**
- Version:  
  4 bits. Protocol version number
- Message Type:
  8 bits:  
  1- Path  
  2- Resv  
  3- Path Error  
  4- Resv Error  
  5- Path Tear  
  6- Resv Tear  
  7- Resv Conf(irmation)

**Object:**
- Session  
  Contains among others, the IP address of the destination.
- RSVP_Hop:
  Contains the IP address of the RSVP-aware router that sent this RSVP message to this node.
- Time_Values:
  Contains the value of the refresh time defined by the source
- Flow Spec.:
  Flow specification specifies the desired service.
- Filter Spec.:
  Filter specification defines which incoming packets belong to this flow.
- Sender_Template:
  Contains among others, the IP address of the source. This is required in the Path packet.

- Adsesc:
Contains some information about the selected path that can be used by the destination. This information could be about the available bandwidth or the maximum delay of the found path. This is optional.

- **Error_Spec:**
  Specifies an error in case of a Path error, Resv error or a confirmation error.

- **Resv_confirmation:**
  Contains the address of the router that has requested a confirmation. This is optional.

Introducing these specifications for the flow brings among others, the following three problems:
1- If an IP packet is fragmented, the structure of the flow specifications will be unreadable for the routers. Some solutions have been recommended for this problem.
2- IPv6 inserts a variable number of variable-length Internet layer headers before the transport header, increasing the difficulty and cost of packet classification or QoS.
3- The security level of IPv4 and IPv6 can hide the flow specifications by encrypting the flow.

### 3.2.2 Path packet

The content of a Path packet is as follows:

- **The address of the previous hop**
  When a Path packet arrives at a router, it contains the address of the previous router. If the traffic control (= admission control + packet classifier + packet scheduler) has accepted the Path packet, this address is saved in the router and the address of the current router is placed in the Path packet and the packet is sent to the next hop.

- **The address of the source**

- **Source_Port**:
  This is the UDP / TCP port from which the data will be sent

- **The address of the destination**

- **Sender Template**:
  This is the filter specification part of the Path packet. The sender template contains information about the packets that will be generated by the source. In
this way the node can select the sender’s packets among the other packets arriving at that node.

- **Sender Tspec:**
  This contains the values of Tspec and Rspec as described in (paragraph 3.2.1).

- **Atdspec:**
  This service is known as “One Pass With Advertising” (OPWA). As there is no end-to-end control of the service by the receiver, this parameter is added to give the destination information about the end-to-end connection set up by RSVP. The traffic control of each router receives this parameter and adds its own information to it. These advertisements may then be used by the receiver to construct, or to dynamically adjust, an appropriate reservation request.

- **Data TTL:**
  This is the IP Time-To-Live. This time defines how long the router should reserve resources according to this Path packet. It is the responsibility of the source and destination to refresh this TTL by sending Path and Resv packets along the path.

- **Policy data:**
  This is optional and identifies certain rights of the source to use some services of the network.

### 3.2.3 Path Error packet

When a next hop can not be found, e.g. because the requirements in the Path packet could not be satisfied, the router informs the source of this fact by sending a Path Error packet back to the source along the same path that the Path packet had followed. As already mentioned, when a router receives a Path packet, it makes a temporary reservation of resources. When a Path packet has been rejected, the previous routers must be informed of this fact in order to release those temporary reservations. The source does this by sending a Path Tear message as soon as it receives a Path Error packet. The Path tear message follows the same way as the Path packet and informs all the intermediate routers to release their temporary reserved resources.

### 3.2.4 Resv Packet

Some elements of a Resv packet:

- The IP address of the next router where the packet should be routed to
- Flow specification
• Tspec, Rspec
• Address of the destination
• Adspec
• Policy data

3.2.5 Resv Error

This is the dual part of a Path error message. If a router cannot find a next hop for the Resv packet, it sends a Resv error message to the destination (the source of the Resv packet) and informs it about the rejection of his request. After receiving a Resv error message, the destination sends a Resv Tear message back through the network along the same path as the Resv packet in order to release the resources that are reserved at the intermediate routers.
This does not mean that the reserved resources at every node will now be cancelled! If a router has merged two or more flows and made one reservation for them all, it cannot delete that reservation by receiving a Resv tear message from one of those flows. This router will discard the Resv tear message and will not send it to the next hop.

3.2.6 Packet scheduler

After the packets are sorted according to which flow they belong, the packet scheduler is concerned with queuing these packets at the output links. To do this, the packet scheduler checks the available queue space left, the priority of the packets and some other parameters. These parameters are then used in an algorithm to decide which packets should be sent to the output queues or should even be discarded.
There are different scheduling algorithms that can be used by a packet scheduler. Two of these queuing algorithms will be discussed here.

3.2.6.1 Class based queuing

The basic idea of CBQ is to share the link capacity between several classes of users [7]. These classes are defined by the user itself or by the Internet connection provider. This classification could be made per application, per flow, per IP source address, etc. For example, it can be decided that 20% of the link capacity is reserved for best-effort services, 30% for real time traffic with priority 2 and 50% for real time traffic with priority 1, where priority 1 is higher than priority 2.
Each of these classes can again be divided into smaller sub-classes, which eventually will result in a tree structure of resource sharing. The amount of capacity assigned to a class defines the minimum capacity received by this class. If a certain class or sub-class exceeds the amount of capacity assigned to it, it can ‘borrow’ some capacity from other classes or sub-classes, assuming that there is enough free capacity and that the installed capacity-sharing-policy of that link allows it. If ‘borrowing’ is not permitted, then the extra packets of that class can be degraded to the best effort class or even discarded.
3.2.6.2 Weighted fair queuing

Weighted Fair Queuing, WFQ, gives the smaller flows (in terms of bytes/s) a preferential treatment and tries to allocate the remaining bandwidth between the larger flows as fairly as possible. First the flows are sorted according to the priority class they belong to and then the flows will be sorted again according to their required bandwidth. The fairness of WFQ is that it protects the smaller flows of being forced aside by larger flows in the network. To do this, it is important to choose an optimal queue length and the right queuing policy. These two can be set by the network operator by monitoring the characteristics of the traffic through his network. Also with WFQ, if a flow exceeds its allowed bandwidth, its packet could be discarded or degraded to best effort. Another option with WFQ is to reshape the flow by introducing certain delays between packets in order to keep the flow characteristics within the allowed range.

3.2.7 Routing

RSVP is not a routing protocol. RSVP is a tool to deliver QoS along the path found by the routing protocol.

If a Path packet or a Resv packet cannot be routed, it will result in a Path error or a Resv error message.
If a Path error, a Resv error, a Path tear or a Resv tear message cannot be delivered, the TTL of a reservation will be ended and the reservation will automatically be released by the router.
Another issue worth of mentioning here is that in this way RSVP can act dynamically and find another path while the flow is going through the network. The flow can then go through this new path and the old path will be deleted once its TTL has ended.

3.2.7.1 Routing through RSVP-unaware nodes

It is impossible to introduce RSVP at the same moment throughout the entire Internet. Furthermore RSVP may never be deployed everywhere, because it does not scale well as it keeps a great deal of flow state information. For these reasons, it is very important that RSVP can operate correctly when two RSVP-aware routers must communicate through a RSVP-unaware set of routers.
Suppose that there are two RSVP-aware routers, router1 and router2, which must communicate through a RSVP-unaware ‘cloud’. Figure 3.1 gives a representation of this situation.
Figure 3.1. A RSVP-unaware ‘cloud’ between two RSVP-aware routers that want to communicate.

When a Path packet is sent out by router1, it carries the address of router1 and the address of its destination. Within the ‘cloud’, the Path packet is routed to the destination as if it is a normal IP packet. Once the packet reaches a RSVP-aware router out of this ‘cloud’ (in this case router2) it informs this router about the address of the previous RSVP-aware router (in this case router1). When router2 wants to send a Resv packet to the source, it sends the packet with the address of router1 through the ‘cloud’. In this case, no guaranties can be given for the service received by the flow, but if the network there has sufficient capacity, it may still provide useful real-time services.

3.2.8 Aggregation of flows

When a router receives several reservation requests, it can decide whether to merge them or not according to the kind of service they require. This aggregation of flows can be very useful during a multicast session, when several users request a connection with the same source and require approximately the same services.

There are four types of reservations within IntServ. Table 1 gives an overview of these types:

<table>
<thead>
<tr>
<th>Sender Selection</th>
<th>Reservations</th>
<th>Shared</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Distinct</td>
<td></td>
</tr>
<tr>
<td>Explicit</td>
<td>Fixed-Filter style (FF)</td>
<td>Shared-Explicit style (SE)</td>
</tr>
<tr>
<td>Wildcard</td>
<td>(Not defined)</td>
<td>Wildcard-Filter style (WF)</td>
</tr>
</tbody>
</table>

In case of explicit sender, each filter specification must match exactly one sender.

- **Fixed-Filter:**
This is a flow with an explicit sender and a distinct reservation requirement. When a router receives such a flow, it will not merge it with other flows and makes an explicit reservation for that flow alone.

- **Shared-Explicit:**
The flows are merged, but there is a list of the explicit senders. This can be used in multicast sessions, where there is one source sending different flows to different users.

In case of Wildcard, no filter specification is needed.

- **Wildcard-Filter:**
The flows are merged, but they are not selected individually, because the flows carry no explicit filter specification! All the senders share the flow specification specified by the source. This can be used in multicast sessions, where a source sends the same flow to different receivers (e.g., a TV multicast application).
WF reservation may be thought of as a shared "pipe", whose "size" is equal to the largest of the resource requests from all receivers, independent of the number of senders using it. Determining the largest request is not always an easy task. One request can be expressed in terms of bandwidth and the second in terms of delay. There are some algorithms in order to specify a certain largest request in these cases, which are out of the scope of this thesis.

The shared reservation made by WF and SE styles are appropriate for use in those communications, where the sender and receiver are not sending packets at the same time. This is because of the maximum resources reserved in case of WF or SE, which is the maximum of all the incoming requests. So WF and SE are suitable for e.g. audio conferences, where the senders do not talk simultaneously during a session. On the other hand, the FF style, which creates a distinct reservation for the flows from different senders, is appropriate for video signals.

RSVP disallows the aggregation of distinct and shared reservations. Aggregation of the explicit and wildcard sender selection is also not permitted. In this way WF, SE and FF are all mutually incompatible.

It is possible to simulate a WF style by using a SE style reservation. We can define a whole list of explicit users, which will be selected by the packet classifier according to their filter specifications. However, when there is a long list of users, the overhead of SE is much larger than an equivalent WF style.

### 3.2.9 Confirmation

A source may ask the network to inform him if enough resources have been reserved for its flow. In two situations, a confirmation message will be sent back to the source:
1. If the reservation has been successful till the last router in the calculated path between the source and the destination.
2. If the reservation comes to a node, where an existing reservation is equal or greater than that being requested. If the two requests can be merged, the router will merge them. Because the original reservation message has now been terminated and is not sent further, a confirmation message is sent back to the source.

Note that the receipt of a confirmation is only a high-probability indication, not a guarantee that the requested service is in place all the way to the sender(s). If the request has been rejected by the network, a confirmation error message is sent back to the destination, by the router that has blocked the flow reservation.
3.3 Some weaknesses of IntServ

The routers keep information about the state of every flow that is being handled by them. The IntServ solutions therefore do not scale well as the number of users, i.e. flows, keeps growing.

If a path refresh packet is lost in the network, the path will be terminated while the flow is still alive.

During a session, the path of a flow may be changed several times. Although no errors may occur in the flow, the network operator cannot optimise his network for a good performance.

There must be a good coordination between the Internet Service Providers in order to make their routers, RSVP-aware. Because if only one ISP implements RSVP on his routers, still no real resource reservation can be made along the path to the receiver.

Finding a path is a very important issue for delivering QoS, but IntServ does not refer to this aspect and relies fully on the routing algorithms installed on the routers. The current routing algorithms of the routers select a next hop only based on the destination address of the packets. This has a negative effect on the performance of IntServ. Using a QoS-aware routing algorithm can improve the performance of IntServ.
4. Differentiated Services

As mentioned, IntServ has the disadvantage that the state of every flow must be maintained by the routers along the path. Trying to solve this problem has led to the definition of Differentiated Services (DS) RFC 2475. In the DS Internet architecture different fixed classes can be defined, each with a specific packet forwarding behaviour. Every incoming packet will be mapped into one of these classes. In this way, a router should only maintain the existing classes and not the state of every flow being handled.

A network operator is usually responsible for the maintenance of several routers. After installing some DS features on the nodes, this group of routers is called a DS domain. A DS domain is a contiguous set of DS nodes, which operate with a common service provisioning policy and set of PHB groups implemented on each node.

In a DS domain there are two kinds of nodes:
1. Interior nodes, which have as their neighbours, only the routers belonging to the same DS domain.
2. Boundary nodes, who have as their neighbours, the routers belonging to the same DS domain as well as the routers not belonging to the same DS domain. A boundary node is called an Ingress node, when it is acting like a port for the incoming packets from other domains and it is called an Egress node, when it is acting like a port for the outgoing packets to other domains.

Figure 4.1 shows a DS domain with different routers within.

![Figure 4.1. A graphical overview of the routers in a DS domain](image)

The PHB defines how certain packets will be handled and forwarded by the router, regarding the assigned bandwidth, delay, etc. To deliver QoS, several PHBs should be defined, each for a certain kind of class. A PHB group refers to the collection of one or more PHBs on a router. The PHB groups are set within every router according to the need of the users of that part of the network or according to the will of the network provider, who tries to optimise the performance of his network. After monitoring the network, the operator may want to change certain parameters of the PHB, which can be done during the operation of the network.
The interior nodes use the DS codepoint of the packet for mapping it to a PHB supported by one of the PHB groups within the domain. This DS field is actually the same as the TOS (Type Of Service) byte in the IPv4 header and the Traffic Class Octet in the IPv6 header. The DS field of a packet has the following form:

![Diagram of the DS-Field of a packet]

The first 3 bits define the class of the packet. As an example can be mentioned the DS codepoint value “000000”, which stands for best effort service. More information can be found in paragraph 4.4.2 and [8], [9].

Although there are some PHBs defined globally, their detailed settings are left to the operator to set. In this way, different DS domains may have different PHB and local DS fields. When the packets leave a DS domain and enter another one, this new DS domain scans the requirements of these incoming packets and maps them into a correct PHB supported by its routers.

### 4.1 Service Level Agreement

A DS region is a set of one or more contiguous DS domains, supporting differentiated services along the path within the coverage of the region. Different DS domains within the region may have different PHB groups and different codepoints->PHB mappings.

To permit services across the domain, the peering domains must each establish a peering Service Level Agreement (SLA). SLA defines how the traffic from one domain to the other should be conditioned at the boundary nodes between the two domains. The two types of SLA are:

1. Static SLA: is agreed in advance between the customer and provider and does not change too much.
2. Dynamic SLA: may respond to variations in offered traffic load and may change frequently. The main issue is how a dynamic control system should be implemented and how the signalling must be done.
Having non-DS-compliant routers in a domain, may result in unpredictable performance and may disrupt the ability to satisfy the service level agreements (SLA).

A SLA defines a Traffic Conditioning Agreement (TCA), which specifies the rules for handling the traffic in terms of traffic profile, metering, marking, discarding and/or shaping, which will be applied to the traffic streams selected by the classifier.

A SLA may also specify packet classification and re-marking rules and may also specify traffic profiles and actions to traffic streams, which are in- or out-of-profile.

4.2 Packet classifier and traffic conditioner

Figure 4.3 shows the architecture of a router in a DS domain [10].

A node consists of the following components:
Packet classifier
Traffic conditioner, which in turn consists of:
   Meter
   Packet Marker
   Shaper / Dropper

The following paragraphs will discuss these components.

4.2.1 Packet classifier

This component may only be installed at the boundary nodes and should not be installed at the interior nodes.
When the packets enter an ingress node, the packet classifier separates the traffic stream into different sub-flows that are to receive a particular service, based on the header of the packet (see figure 4.4) [11].
There are two types of classification:
1. Behaviour aggregate classifier: uses only the DS field in the IP header of the packet. This is very simple because of the 1-byte field of the DS field.
2. Multi-Field classifier: uses a combination of one or more header field such as source address, destination address, DS field and some other information. A disadvantage of this classifier is its complexity.

Traffic conditioner ensures that the traffic entering the DS domain is conform the rules specified in TCA. The complexity of the traffic conditioner depends on the extent of services of the DS domain. It could be a simple operation of modifying the DS codepoints of the packets or a more complex operation of modifying the DS codepoints and reshaping the traffic.
Traffic conditioners are most useful in ingress or egress nodes, because once they have applied the TCA on the traffic, the interior nodes do not need to change this traffic again. However, the traffic conditioners can exist at traffic sources at interior DS nodes as well. In a particular node in a DS domain, some or even all of the functions shown in the traffic conditioner may be null.

4.2.2 Meter

A meter measures the traffic stream selected by the classifier against a traffic profile specified in a TCA. The meter can then identify the packets as in-profile or out-of-profile. The in-profile packets will be send further and the out-of-profile packets are discarded or handled in a different way than the service they required. A meter can choose the state of the packet in accordance to several algorithms:
1. Average rate meter
2. Exponential weighted moving average (EWMA) meter
3. Two parameter Token Bucket meter
4. Multi-state Token Bucket meter

Average rate meter

An average traffic ($\alpha$ bits/s) and a delta period ($\Delta$ msec) are defined. The meter measures the overall traffic of the packets between $T$ and $T-\Delta$ msec. If the traffic in $\Delta$ msec does not exceed $\alpha$, the traffic is marked as in-profile and the traffic is marked out-of-profile otherwise.

Exponential Weighted Moving Average (EWMA)
The following formula is used:

\[ \text{avg}_\text{rate}(t) = (1 - \text{gain}) \times \text{avg}\_\text{rate}(t') + \text{gain} \times \text{rate}(t), \]

where
\[
t = t' + \Delta
\]
\[
\Delta = \text{a small fixed sampling interval}
\]
gain controls the time constant of the response
\[
\text{rate}(t) \text{ measures the number of incoming bytes during the time interval } \Delta
\]
The parameter AverageRate is also pre-defined by the SLA.
The packets are marked as in-profile if \((\text{avg}\_\text{rate} < \text{AverageRate})\) and out-of-profile otherwise.

Two-parameter Token bucket meter

An average rate \((\alpha \text{ bits/s})\) and a burst size \((\beta \text{ bytes})\) are defined. \(B\) is the total number of tokens and \(\alpha\) is the average rate of the incoming tokens. An arriving packet is being handled only if there is a token in the buffer. When there are no tokens in the buffer, that arriving packet at that moment is identified as out-of-profile. \(B\) is the burst size because the packets are allowed to exceed the average rate up to the burst size after which there will be no token left.

![Figure 4.5. A graphical overview of the two parameter Token bucket meter](image)

Multi-state Token Bucket meter

This is a cascade implementation of several two-parameter Token Bucket (TB) meters. If the buffer of the first (TB) overflows, the exceeding packets are sent to a second TB buffer. If this buffer also overflows, the exceeding packets are sent to the third TB and so on dependent on how many TB are cascaded. Figure 4.6 shows an implementation with two TB cascaded.
The packets exceeding neither of the buffers are marked as in-profile. The packets exceeding smaller burst size are marked partially in-profile. The packets exceeding larger burst size are marked as out-of-profile.

4.2.3 Marker

A marker contains the relationship between the services in other DS domains and its own DS domain and also the relationship with non-DS domains such as an IntServ domain. According to these relationships, which are specified by SLA/TCA, the DS codepoint of the packets will be modified so the packets can be mapped to the right PHB through the DS domain. In this way also the DS codepoint of the out-of-profile packets is modified to inform the other routes what to do with them. These packets could be degraded in service or dropped.

Shaper

A shaper ensures that a flow conforms exactly to the parameters given by a certain traffic profile (in-profile / out-of-profile). This may cause some packets to be delayed. If the actual packet stream is very different from the traffic profile, this could result in dropped packets, because of the finite buffer of the shaper.

Dropper

As the name suggests, the dropper is responsible for dropping the packets from a flow in order to force conformance with the TCA. The dropper may use the information from the classifier, meter and marker in making its decision about whether to drop a packet. Whether a dropper is in operation for all or part of a flow is determined by the SLA/TCA. Note that a dropper can be implemented as a special case of a shaper by setting the shaper buffer size to zero packets.

4.3 The location of traffic conditioners and classifiers
The traffic conditioner and classifiers are usually located within the ingress / egress nodes, but they can also be located at the interior nodes, sources or even on a non-DS-capable-router.

4.3.1 At the boundary nodes of a DS domain

This may be the most common place for the traffic conditioner and classifier. In this way the complexity of the actions of the interior nodes is reduced. An egress node makes sure that the traffic conforms the SLA/TCA to the peering domain, but the ingress nodes of that peering domain must assume that the oncoming traffic may not conform to the TCA and must be prepared to enforce the TCA in accordance with local policy.

4.3.2 At the interior nodes of a DS domain

Although it is assumed that the complexity of the packet handling should be located only at the boundary nodes, some functionalities could also be deployed at the interior nodes. In this way more restrictive access policies may be enforced, but with the problem of scaling limits, because of the large number of classification and conditioning rules that might need to be maintained. The installed functionalities could also be a simpler version of that of the boundary nodes.

4.3.3 At the source

When a source generates traffic, it can perform classification and conditioning on the traffic directly. This has the advantage that the source can more easily take into account the requirements of an application. A second advantage is that classification and conditioning of packets will be easier, because the packets are not mixed with other packets of other traffics.

The source should be aware of the policies and the TCA in that DS domain, because the disadvantage of classification and conditioning at a source is that a source may try to label all its packets as ‘high priority’ and use the network resources as much as possible.

4.3.4 At non-DS-capable nodes

Classification and conditioning may be deployed on a non-DS-capable node that has a peering connection to a DS-capable node, to pre-mark the traffic before it reaches the ingress node of a DS domain.

4.4 Per hop behaviour

There are Internet Drafts proposing a number of PHBs. In this section the Expedited Forwarding PHB (EF-PHB) and the Assured Forwarding PHB (AF-PHB) group will be discussed.
4.4.1 Expedited Forwarding (EF)

The characteristics of this service are: small packet loss ratio, low delay and delay variation (jitter) and a more or less guaranteed bandwidth. [12]. The idea is that the packets in this service should be forwarded by the node as quickly as possible, regardless of the other packets or traffic handled by that node. There is however a maximum bandwidth that EF PHB is allowed to use, which is set by the network administrator. This maximum bandwidth can be controlled by e.g. a token bucket rate limiter.

Two packet scheduling mechanisms that may implement this service could be: 1) a simple priority queue, 2) Class Based Queuing.

4.4.2 Assured Forwarding group (AF)

AF is a PHB group with 12 different services within. The purpose of this service definition is to allow a domain to provide different levels of delivery assurance. There are 4 AF classes and 3 drop preference level within each class [13] [10].

<table>
<thead>
<tr>
<th>Drop Preference</th>
<th>AF 1</th>
<th>AF 2</th>
<th>AF 3</th>
<th>AF 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low drop preference</td>
<td>010 000</td>
<td>011 000</td>
<td>100 000</td>
<td>101 000</td>
</tr>
<tr>
<td>Medium drop preference</td>
<td>010 010</td>
<td>011 010</td>
<td>100 010</td>
<td>101 010</td>
</tr>
<tr>
<td>High drop preference</td>
<td>010 100</td>
<td>011 100</td>
<td>100 100</td>
<td>101 100</td>
</tr>
</tbody>
</table>

Table 4.1. DS byte codepoints for AF

Table 4.1 shows the codepoint of different AF classes and drop preferences. The six DS bits are divided into two ‘fields’: the first 3 bits are the AF class number with the decimal value of 2, 3, 4 and 5. The second 3 bits are the drop precedence with the decimal value of 0, 2 and 4. The numerical ordering of the class field implies an ordering of the classes. If the numerical class field \((x) < \text{class field (y)}\) then class \((y)\) has at least as many forwarding resources allocated to it as class \((x)\). In this way, AF 1 has the lowest priority. Likewise, the ordering of the drop precedence fields implies an ordering of drop ratios: if precedence fields \((p) < \text{precedence fields (q)}\) then the drop ratio of packets within a AF class with drop precedence \(p\) must be at least as small as the drop ratio of packets marked with drop precedence \(q\).

The flows belonging to different classes are not allowed to aggregate and must be forwarded independently. Furthermore the packets can be dropped by the traffic conditioning actions, but they can also be upgraded or downgraded in drop precedence, or even assigned to another AF class. If packets are assigned to another AF class, packets from the same flow are not allowed to be assigned to different AF classes.

4.4.3 The Class Selector PHB Group
The delay of a packet in a class of CS-PHB group must be not lower than that of a packet in a lower class of CS-PHB group. This goal is called timely forwarding requirement. In this way every node should monitor its amount of resources used, in order to enforce a packet of this group if necessary. Moreover, the CS-PHB may be used to provide a grade of service differentiation in respect to delay of bandwidth. To achieve these two goals simultaneously may result in a conflict. This can be solved by seeing the bandwidth enforcement as a tool for controlling the delay and in this way giving a higher priority to the timely forwarding.
5. Multi Protocol Label Switching

MPLS [14] is a forwarding scheme that adds a label to the incoming packets and subsequently classifies and forwards them according to this label. When a packet enters a MPLS-aware domain, the ingress node adds a label to the packet regardless of the type of the packet. This justifies the name, “Multi protocol”, because its techniques are applicable to any network layer protocol.

In a non-MPLS network when a packet arrives at a router, the router will scan the destination address of the packet and tries to find a match in its routing table and assigns the packet to a particular Forwarding Equivalence Class. A FEC is a stream of packets, which get the same treatment by a router and are sent to the same next hop. This next hop will in turn re-examine the packets and assign them to a FEC.

In MPLS, the assignment of a packet to a FEC is only done by the boundary nodes of the domain. The information about the FEC where the packet is mapped to, will be put in a “label” and attached to the packet. When the packet goes through the network, there is no further analysis of the packet’s network layer header. Rather, the relatively less complex label is used as an index into a table, which specifies the next hop and a new label and then the packet is forwarded to the next hop. This forwarding of a packet, based only on its label, has several advantages over conventional network layer forwarding:

- The forwarding of the MPLS labeled packets can be done by routers that are not capable of analyzing the network layer header, or are not capable of analyzing the network layer header at a reasonable speed.
- When a boundary router assigns a packet to a FEC, it can use more information than only the information carried within the packet. For example, packets arriving from different ports can be assigned to different FECs.
- A packet entering a domain through a certain router can be assigned to a different FEC then the packet arriving at the same domain from another router.
- The rules that describe how a packet is assigned to a FEC can become more and more complex without the need of modifying the routers that merely forward labeled packets.
- The labels of a packet can force the packet to follow a certain path.

Some routers may not only choose to next hop of a packet from its label but also some additional information about its class of service or drop precedence. In this way they can apply different scheduling or discarding algorithms.

5.1 Label assignment and distribution

When the routers Rs and Rr want to set up a packet transition from the sender router Rs to the receiver router Rr, first they agree on a binding of a certain label content (Lc) and a certain FEC (F). When Rs is sending a packet to Rr and that packet belongs to class F, Lc will be assigned to the packet and the packet is sent
further to Rr. This does not mean that Rs labels all its packets belonging to F with the label Lc. Lc is an arbitrary value whose binding to F is local to Rs and Rr. Note that in the above explanation Rs and Rr do not have to be the generator and the consumer of the packets but they also could be two intermediate routers along a path of a packet.

It is also important that Rr knows for sure that the incoming packet is from Rs and not from another router. That is why it is important that Rr does not make the same agreement as with Rs to bind F to the label value Lc.

The decision to bind a particular FEC to a particular label is made by the Rr (the receiver). This router then informs the Rs (sender) of the binding. So the labels are “receiver-assigned”.

It is possible that router 1 receives a FEC/label binding from router 2, even though router 2 is not the next hop of router 1 for that particular FEC. In this case the router may:
Discard the binding. If at a certain moment router 2 becomes the next hop of router 1 for that particular FEC, the binding will have to be re-acquired. In this way the router requires to maintain few labels but the adaptation to routing changes is slow.
If a router supports “Liberal Label Retention Mode”, it maintains all the incoming bindings and uses them when necessary. This result is a quicker adaptation to routing changes but a larger table of bindings that have to be maintained.

A label will be removed from the packet when the packet is leaving the MPLS network.

5.2 Label Distribution Protocols
In order to send their binding decisions to each other, the routers use a Label Distribution protocol (LDP). Two routers that inform each other of their FEC/label bindings are called “label distribution peers”. The architecture of MPLS does not require that there is only one LDP.

5.3 Routing
MPLS supports two kinds of routing methods:
1. Hop by hop routing
2. Explicit routing

5.3.1 Hop by hop
Each router examines the label of the incoming packet and according to its table of bindings, sends the packet to a next hop.

5.3.2 Explicit routing
In some situations it could be desirable to forward the packet through a pre-specified path. This could be done by the network administrator for a better utilization of the network. In this way a “tunnel” is created where certain packets are pushed through. The path may be pre-configured or it may be determined
dynamically by some means e.g., by constraint-based routing. It may also be desirable to apply resource reservation along that path. This can be done e.g. by RSVP.

This ability of MPLS to set up explicit paths is very useful for building up Virtual Private Networks (VPN).

5.4 Invalid labels
When a router receives a packet, it could be possible that the packet is not labeled due to some errors in the previous hop. The policy is to discard these packets, because sending them forward on the basis of their network layer header can cause loops. The routing algorithm may decide based on the network layer header of the packet that it should be sent to a certain next hop, but this next hop could be the very same router from where that packet has come.

5.5 MPLS in a Differentiated Services domain
Because of the fact that MPLS is path-oriented, it can deliver better and more reliable QoS when combined with the connectionless-oriented Differentiated Services. The paths between the boundary nodes of a DS domain can be configured and during the operation the forwarding of packets will be based on their label rather than the DS-field. There are three differences in processing a packet:
1. At the ingress node of the DS domain, a label will be inserted into the packet. The processing in the DS-field will remain unchanged.
2. The interior routers forward the packets according to their labels.
3. At the egress nodes, the MPLS headers will be removed.

Whether a network uses DS-fields or labels to forward the packets is transparent to the network administrators. By making the appropriate SLAs, the neighboring domains can guarantee the interoperability between a DS-field-based and MPLS-based architecture. In this way these two techniques can coexist.
6. QoS Routing in IntServ

The studies on IntServ so far are based on non-QoS-aware routing algorithms. The advantage of using a QoS routing protocol is the ability of taking into account two or more constraints (e.g., bandwidth, delay, number of hops, etc.). The combination of a QoS Internet architecture and a QoS routing algorithm could be a very powerful mean to optimize the performance of the Internet and the applied architectures.

Simulation has been used as the main tool for studying the effects of implementation of a QoS routing algorithm in the IntServ architecture. Paragraph 6.1 will first give a brief introduction to Samcra followed by paragraph 6.2, where the simulator that is used for this project is explained. This introduction will be brief, for a detailed manual we refer to [15], which was written as a part of this thesis.

6.1 QoS routing algorithm, Samcra

Samcra [1] [16] is a constraint-based routing algorithm, which means that it tries to find a path on which the values of the metrics remain below a specified set of constraints. The metrics should be additive, which means that their value can be added along the path. In this definition, delay is an additive constraint and bandwidth is not.

If there exists a path that satisfies the constraints, it is proved that Samcra will find that path [1]. If such a path does not exist, of course no path will be returned.

Part of the motivation of this thesis is the expectation that integrating Samcra with IntServ, can improve the performance of the network in comparison with the case, where a non-QoS routing protocol is used.

6.2 QoS Routing Simulator

QoS Routing Simulator (QRS) [17] is a packet-level routing simulator based on the well-known MARS (MAryland Routing Simulator) [18]. The term ‘packet-level’ means that the simulator simulates a network on the level of packets. At the end of a simulation, the behavior of the network in terms of packets could be monitored. A lower level simulator may simulate a network on bit-level and a higher level simulator may simulate a network on flow-level. Of course, the higher the level, the less complex the simulator will be. Although MARS has a graphical interface, QRS is a totally text based simulator, where the input as well as the output are in text format. For this thesis, version 1.1 of this program has been used, which only supports IntServ.

The two most important aspects of the program for this thesis are the routing algorithms and the link state update policies implemented.
6.2.1 Routing

There are two routing algorithms available in QRS1.1, the lowest cost algorithm and the widest bandwidth algorithm. The user can choose the desirable algorithm. The lowest cost algorithm finds the path with the minimum cost, where the definition of the cost can be chosen by the user from one of these options:
1. Delay
2. Hop
3. Hop normalized delay
4. Utilization

The widest bandwidth algorithm finds the path with the largest available bandwidth. If there are two paths with the same bandwidth, the one with the smaller delay is chosen.

Samcra is implemented in QRS 1.1 as the third routing algorithm. A hop-by-hop version and an explicit routing version are now available.

6.2.2 Link state update policy

In order to calculate a path for a packet and to send it to the right next hop, every router should have some kind of topology information of the network, where it is a part of. Moreover, during the operation of the network, every router should have updated information about the availability of certain links or nodes and their conditions. This is done by flooding. Flooding is the process where every router broadcasts its link information to its neighbor nodes. This information could be the address of its neighbor routers or the available bandwidth on its links. By receiving this information, the neighboring nodes send this information plus their own state to their neighboring nodes and so on. In this way the routers will have a realistic and updated view of the network. The question that rises here is ‘when to broadcast this information?’ The policy, which triggers this broadcasting, is called ‘Link State Update policy’. LSUp determines when a significant change in the links of a router has occurred that is worth of broadcasting.

There are two major issues when considering the LSUp:
1- The information that a router broadcasts to other routers is encapsulated in packets. Sending these packets means using the available link bandwidth. So the more often a router floods, the more network resources will be utilized, but also the more accurate the information of the router will be regarding the network, which makes it possible to find better paths. This is a clear case of trade off.

2- The information that a router receives must be processed in the router. The router may change its network topology and recalculate the routing table. This is a very time consuming process during which the router is unable to handle the packets. This is again a trade off between fewer calculations but also less updated information or more calculations and also better-updated information.
These two points form enough reasons for a wide area of research in order to develop more intelligent LSUp to optimize the flooding.

There are 4 types of link state update policies within QRS1.1:

1. **Period based (PB)**
   Flooding happens periodically, where the value of the period can be set by the user.

2. **Threshold based (TB)**
   The following formula is used:
   \[
   \frac{B_{la} - B_{ca}}{B_{la}} \geq \text{Th}, \quad \text{Where:}
   \]
   
   - \(B_{la}\) = Last Available Bandwidth
   - \(B_{ca}\) = Current Available Bandwidth
   - \(\text{Th}\) = Threshold value (in percents)

   If the value of the left side of the equation is bigger than the value of the threshold, which is set by the user, LSUp will trigger a flooding event.

3. **Equal class based (ECB)**
   The bandwidth of the link is divided into a number of classes, which can be set by the user: \((0,B), (B,2B), (2B,3B), \ldots\) etc, where \(B\) is the size of one class.
   The router will flood, every time that the utilization of the link passes the border of two classes.
   In figure 6.1 the available bandwidth of a link is divided into 3 classes. The figure shows the link utilization with respect to time. The points A, B, C and D are four examples of the points where the router will broadcast its link states. Between these points the utilization remains in the same class and flooding is not triggered.

4. **Unequal class based (UCB)**
   The idea of the UCB is basically the same as ECB with the difference that the classes do not have the same size. The two parameters that define an UCB are \(B\), which is the size of the smallest class at the top and \(f > 1\). The class distribution is according to:
   \((0,B), (B, (f+1)B), ((f+1)B, (f^2+f+1)B), \ldots\) etc.
When the link utilization has nearly reached the maximum capacity of the link, it becomes crucial to update the link state more regularly to avoid over-utilization, which may result in packet loss. This is the reason that at higher link utilizations, the classes are smaller.

Note that ECB is a special case of UCB, which can be achieved by setting the value of ‘f’ to ‘1’.

Besides these 4 LSUp, there is also a mechanism that is used in combination with a LSUp to reduce the costs of flooding (cost in terms of network overhead and cpu time). This mechanism is called Hold-timer defined in seconds. The hold-timer is used, when the LSU too often triggers flooding. This could be caused by the variations in the link utilization. By defining a hold-timer of $x$ seconds, the LSUp will not trigger flooding before at least $x$ seconds have passed from the last flooding event. In this way the flooding cost is bounded, but because of the trade off, the network information at the routers will be less accurate.
7. Simulation results

This chapter discusses the simulations, which are the basics of our conclusions. Paragraph 7.1 presents our simulation plan. The results and the accompanying explanations are given in 7.2.

7.1 Simulation plan

7.1.1 Topology

The topology used for the simulations is a backbone in North America. This topology is chosen, because it is a realistic network. Although this network does not contain many nodes, it has been chosen because of the limited number of 50 nodes allowed by the simulator. By assuming that every node represents one or more ISP networks, the results of the simulations will have more practical value.

![Topology diagram]

Figure 7.1. The topology used for the first simulation set

7.1.2 Source / destinations

30 sources of equal priority have been defined in the network with a flow rate of 512 KB/s each. In this way every node has been attached to at least one sender and one receiver. Having only 30 flows in a network with 28 nodes is not enough, but this is the maximum number of flows allowed by this version of the simulator. It is also not realistic to have a flow that consumes one third of the total link bandwidth. The reason for having such large flows is a very low network utilization that will be achieved otherwise, again because of the limited number of flows allowed by the simulator. The average link utilization during this simulation session is approximately 40 %.

The flows are all real time, which in this case means that RSVP has been used to reserve resources. FTP or Telnet sources have not been used because of the pollution they would introduce in the simulation results. Because of the characteristics of TCP, these sources will only send if there is enough bandwidth available and will not send or send less otherwise. Introducing such sources, will not only provide no additional value to the simulation results, but the results will even be polluted by the behavior of TCP, which will try to fill up the empty space available on a link, which in turn results in always-highly utilized links. This will prevent a good study of flow dynamics in a network.
7.1.3 Links

There are 45 links with a capacity of 1.5 Mbytes/s each. A higher link capacity is not chosen. The following two reasons can be mentioned for this decision:

1. By making the link bandwidth larger with a factor 2, the flow rate of the source should also be made larger with a factor 2 in order to achieve a reasonable link utilization. So the proportional relation between the flow rate of the source and the link bandwidth is the important parameter in this simulation.

2. Making the flow rates larger, e.g. ±13 (= 155M / 8 / 1.5M) times larger, the simulation time would also become larger, because of the generation of 13 times more packets.

During both simulation sets, it has been assumed that a link or a node will never fail, as the probability of failing of a link or a router seems to be very low.

7.1.4 Simulation time

For choosing the right simulation time, the two following points should be considered:

1. In QRS 1.1 the flow rates of the sources have certain distributions but their average behavior in a large time scale does not vary much.
2. When starting a simulation, the network is not assumed to have an instantaneous utilization on every link. A certain time must elapse before all the links in the network have reached a certain utilization.

Regarding these two points, it can be stated that choosing a very short simulation time does not let the network reach its equilibrium point and choosing a very large simulation time will not add any extra information because of the relatively ‘constant’ state of the network.

For this purpose the following plot has been made. In figure 7.2, the total network throughput is plotted with respect to time. Total network throughput is the total number of data bytes that have been acknowledged during a certain interval divided by the length of that interval [18].
It can be seen that after about 40 seconds the network has reached a stable state of operation. Considering this result, a simulation time of 100 seconds has been chosen to give the network enough time to reach its stable point and operate long enough in that state.
Note that with 100 seconds, is meant ‘100 simulation seconds’, which could be shorter or longer than 100 real seconds depending on the size of the network, the number of flows, their flow rates, the chosen LSUp, etc.

7.1.5 The input parameters

The parameters that have been changed for the simulations are:

- Type of routing algorithm
  1) Lowest cost (LC)
  2) Samcra hop by hop (Samcra hbh)
  3) Samcra Explicit Routing (Samcra ER)
- Type of LSUp with their accompanying parameters
  1) Period based (PB)
  2) Threshold based (TB)
  3) Hold timer (HT)

Table 7.1 shows the combination of these two parameters that have been simulated.

<table>
<thead>
<tr>
<th>LSUp</th>
<th>Routing algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LC</td>
</tr>
<tr>
<td>PB</td>
<td>TB</td>
</tr>
<tr>
<td>HT</td>
<td>HT</td>
</tr>
</tbody>
</table>

Table 7.1. The combination of routing algorithm and LSUp used for the simulations.
Widest bandwidth could not be used for our simulations, due to an unsolved bug either in QRS1.1 or in the network configuration file.

The values of PB LSU have been set to:
0.1, .05, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (s)
**Motivation:** Update values larger than 10 seconds result in unrealistic and unpredictable behavior of the simulator. These values are also used in other reports [19], where QRS1.1 has been used.

The values of the TB LSU have been set to:
10, 20, 30, 40, 50, 60, 70, 80, 90, 100 (%)
**Motivation:** a value smaller than 10% will result in too much flooding events and a value larger of 100% is too large for triggering flooding. With these values the whole range of possible values can be monitored.

The values of the HT have been set to:
0.1, .05, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (s)
**Motivation:** These values have been chosen equal to those of PB, because of the expectation that introducing a hold timer will result in a PB-like behavior. Introducing a HT value has of course no meaning when using PB policy.

*Equal class based and Unequal class based LSUp are intentionally not used.*
**Motivation:** In our model, there can be a maximum of 3 flows in one link with the consequence that defining more than 3 classes for LSUp will be meaningless! If there are more than 3 classes, some of them will never be used during a simulation. This assumption is verified by a simulation. The result can be seen in figure 7.3.

![Equal classes policy](image)

Figure 7.3. Total network throughput with respect to the Number of classes in with ECB LSUp

38
It is clear that in the case that the link bandwidth is divided into more than 3 classes, the total network throughput will not be affected anymore. Having 3 or less classes is an unrealistic situation with no added value for a study.

**Cost definition:** The lowest cost algorithm minimizes a certain cost after filtering for bandwidth, which can be defined by the user. This cost will also be given to Samcra as one of the metrics.

During the simulations, the following definitions for the cost have been chosen:

- **Delay:** Represents the average delay suffered by a packet on the outgoing link queue and in transmission (including the processing time at the node and the propagation delay of the link), over all packets transmitted in the last link-cost update period.

- **Hop-normalized delay:** The average queuing delay per packet and the average transmission delay per packet are measured over the last link-cost update period. The utilization is calculated from these delays, assuming an M/M/1 queuing model. The main purpose of hop-normalized-delay is to make the links that were highly utilized during the last link-cost update period more attractive to the packets than the situation where ‘delay’ is used as the cost. In this way a better throughput and load balancing may be achieved.

- **Utilization:** Fraction of the time that the outgoing link queue (including the packet being transmitted) was not empty in the last link-cost update period.

**Constraints:** All the simulation have been done for 2 sets of constraints, which were (infinite, infinite) and (9,4). The first constraint is for the cost and the second is for the number of hops. The value of 9 is chosen as an upper bound for the cost after examining the simulation results on the delay per packet. 9 is lower than the average delay of a packet but yet not too low to interrupt the network operation. The value of 4 is more or less the average value of the number of hops of a flow in this network. This has been determined after some simulations.

### 7.1.6 The output parameters

There are many output parameters in QRS1.1 that can be monitored. For choosing the right parameters, the focus points of this thesis should be considered. We study the effect of routing protocols and LSU policies on the performance of a network. Calculating different paths by different routing algorithms will result in differences in the total amount of data sent through the network, the total cost along the path (e.g. delay), number of packets dropped, the number of IntServ flows being blocked by the network or the time consumed by the routers to calculate the path and “level of load-balancing”.

The effect of having different LSU policies may be seen in a change in the total amount of data sent through the network or the total time consumed by the routers for flooding and updating link states and flow tables.

These considerations have led to monitoring the following parameters:
• Total network throughput: the total number of data bytes that have been acknowledged during a certain interval divided by the length of that interval
• Average delay per packet: The delay of a data packet is the time difference between the first time the data packet is sent by the source into the network and the receipt of its corresponding ack packet at the source.
• Total time consumed in QOSPF: this time represents the total time consumed by routing calculations + link state broadcasting during a simulation. QOSPF is the name of the component responsible for routing and link state update.

It was also desirable to monitor the values of total number of dropped packets and the total number of blocked flows. Under the simulation conditions in this set, the total number of dropped packets remained zero. This number could increase by increasing the flow rate of the sources, but as they were already large (500Kbytes/s), the number of dropped packets is not monitored in this set of simulations.
Furthermore, QRS1.1 provides some means to monitor the total number of blocked flows. During this project, much effort has been made to make this tool operational, but unfortunately without success.
7.2 Simulation results

In this paragraph the simulation results will be presented and discussed. The first two paragraphs discuss the network behaviour when respectively PB and TB policy are used. Paragraphs 7.2.3 and 7.2.4 present some conclusions that are related to the simulation results and finally the last paragraph explains the implementation of the services within IntServ and how our conclusions can contribute to some recommendations for implementing these services.

7.2.1 Period based update policy

In this paragraph and in the following, first the results are discussed for the simulations where the values of the constraints have been set to infinity. This is indicated by the sub title ‘No constraints’. This is followed by an explanation of the case that the constraints have a finite value.

No constraints

For all the combinations of routing algorithms and different cost definitions, the total network throughput has been plotted with respect to the values of the period update.

![Graph showing total network throughput vs PB value](image)

Figure 7.4. The throughput of the three routing algorithms with respect to the update value. The cost is defined as ‘hop-normalized-delay’.

In all the figures of this report, a dot in the plot represents one simulation of 100 seconds. So in figure 7.4 every line is made out of 12 simulations, where the value of the update period during that simulation is equal to the corresponding value on the x-axis.

The simulations show a waterfall effect for the three routing algorithms. For lower values of the update period, the routers contain updated information about the state of the network and can calculate optimised paths. The waterfall effect
appears when the update period becomes larger and the information at the routers becomes less updated. The calculated paths are then not optimised, which results in less throughput.

The performances of the 3 routing algorithms for different cost definitions are very much the same in terms of throughput, average delay per packet and the total cost in QOSPF.

This common behaviour can be explained as follows:
The packet delay measured on a link has three components: queuing delay, transmission delay and propagation delay. Of these three, only queuing delay depends on the utilization of the link.

When the network is highly utilized, queuing delay will dominate the transmission and propagation delay. In this way the cost definitions of ‘utilization’ and ‘delay’ will approach each other in value. Also ‘hop-normalized-delay’ is defined to be sensitive for queuing delay at high network utilization [20]. At low network utilization, ‘hop-normalized-delay’ and ‘delay’ are dominated by the propagation and transmission delay and apparently in our network, the queuing delay under low utilization shows a great comparison to the transmission delay. An explanation for this could be the relative small network used for the simulations, where the number of nodes and flows is limited.

Samcra ER shows a relative insensitiveness to the value of the update period. This behaviour can be studied using figure 7.5 and 7.6, where each bar represents the utilization of a certain link. So figure 7.5 shows that the utilization of the 20th link is about 20% and the utilization of the 40th link is zero. There are 45 links in the network. Figure 7.5 shows the link utilizations of Samcra hbh algorithm with a periodic update value of 9s and figure 7.6 show the same but for Samcra ER.
Figure 7.6. Link utilizations of Samcra ER

It is easy to see that Samcra ER has a much better load balancing, where many links are used with low utilizations versus Samcra hbh, where few links are used. The lowest cost algorithm shows the same performance as Samcra hbh. This is the reason why Samcra ER has a higher throughput.

But we must explain why Samcra ER has a better load balancing than the other two algorithms.
First we should note the following:

- In our simulations, the PB update value has a range from 0.1s to 10s.
  In our relatively small network, the average number of nodes along a path is about 5 and the time needed to pass these nodes is in the extend of milli-seconds. Taking this into account, it can be concluded that at the time of the generation of a flow and during the reservation of resources along the path for this flow, the source has the same updated information available as all the other nodes along the path. So in our case, the path calculated by Samcra ER does not get out of date during its setup.

- Second, it is proven that a hop by hop calculated path could be different than an ER calculated one [16]. Figure 7.7 show such a situation. The number above a link, are the values of the metrics on that link. A path must be calculated from node 1 to node 3.

Figure 7.7. A part of a network with 3 nodes.
Samcra adds the metrics along both paths \{(11,6), (12, 5)\}, takes the minimum of the two maximum values of these sums and chooses the path with the minimum value \{\min (\max (11,6), \max (12, 5))) = 11\}. In this way node 1 calculates the path along link1 and link2. But when the flow arrives at node 2, it will be routed through link 3, because link 3 has the smallest value of the metrics \{\min (\max (1,3), \max (2,2)) = 2\}.

The above example shows that the path calculated by the source could be different from the path found in a hop-by-hop manner along the path. It can also be seen that the path calculated by node 1 has a better total performance than that found by node 2. The conclusion is that a source-based calculated path could be more optimised than the hop-by-hop path. So in figure 7.7 the source has calculated an optimised path, but due to the hop-by-hop behaviour, this path has been changed.

Summarising the above mentioned points we can note that:

- When the update period is small enough, the effect of availability of very updated information at the routers dominates the effect of calculating the paths source-based.
- When the update period is large enough, the effect of calculating the paths source-based dominates the behaviour of the network because of the lack of updated information at the routers.

The results of average delay per packet are strongly related to the total network throughput. The more packets in the network, the more loaded the network is and so the bigger the delay in handling the packets. It can be seen that, because of a relative constant throughput value of Samcra ER, its average delay per packet is also fairly constant.

![Figure 7.8. Average delay / packet of the three routing algorithms in respect to the update value. The cost is defined as ‘hop-normalized-delay’.

```text

<table>
<thead>
<tr>
<th>PB value (s)</th>
<th>Average delay / packet (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
</tr>
</tbody>
</table>
```

---
For small values of update period, the average delay per packet of Samcra hbh is the highest, because its throughput is also the highest (see figure 7.4). It is also interesting to note that Samcra ER has the lowest delay for small values of update period, although its throughput is higher than that of the lowest cost algorithm. This indicates that for small update periods, the paths found by Samcra ER are more optimised in terms of delay than the other two algorithms. This is due to the fact that hop-by-hop routing may result in less optimised paths than explicit routing, as explained before.

After the update period has become large enough, the lowest cost algorithm handles the packets with the least delay, because of the decrease in its throughput. The delay of Samcra hbh tends to adopt the behaviour of Samcra ER, although the throughput of Samcra hbh is less than that of Samcra ER. This relative high delay per packet of Samcra hbh can be explained by referring to the explanation above, about the behaviour of a hop-by-hop routing algorithm. Apparently, because of the lack of updated information at the routers, the paths calculated by Samcra hbh are not optimised in terms of delay and load balancing and therefore the delay per packet is high even when the throughput is low. In the same way, the paths calculated by Samcra ER are optimised and therefore the delay per packet is low even when the throughput is relatively high.

The time consumed in the QOSPF component decreases with increasing update period. At a large update period, the routers will loose less time for broadcasting and flow table updating, which results in a reduced cost in the QOSPF component.

Constraint (9, 4)

The meaning of the constraints is that the found path must not exceed the value 9, in terms of cost and must not exceed the value 4 in terms of number of hops along the calculated path.
Introducing these constraints forms a restriction on the path found by Samcra. If the constraints are small, they result in fewer paths that satisfy the constraints, which in turn results in less throughput.
It is noticeable that in spite of a decrease in the case that ‘utilization’ or the ‘hop-normalized-delay’ is defined as the cost, the throughput is relatively high, while the delay remains under control. Both cost definitions show a great resemblance.

A remarkable point is that for update period values smaller than 4 seconds the average delay per packet of the lowest cost algorithm (figure 7.8) is larger the average delay per packet of Samcra with constraints (figure 7.10). The corresponding throughput of lowest cost algorithm is between 6000 and 8000 Kbytes/s as can be seen in figure 7.4, while the throughput of Samcra is approximately 7000 Kbytes/s (figure 7.9). This implies that for update values smaller than 4 seconds, Samcra can deliver approximately the same throughput, while the delay is less or equal to the lowest cost algorithm.
An explanation for this behaviour could be the fact that by introducing constraints, the value of the costs based on ‘utilization’ and ‘hop-normalized-delay’ are less than that based on ‘delay’. In this way Samcra is more likely to find a path that satisfies the constraints.

• The cost definition of ‘utilization’ is strongly related to the queuing delay at the buffers. By introducing constraints, the number of paths that satisfy the QoS requirements will decrease and so the throughput decreases. A decrease in the throughput means less occupied buffers, which results in a smaller value of the cost, based on ‘utilization’.

• The effect of ‘hop-normalized-delay’ is to make routing reasonably sensitive to the propagation, transmission and queuing delays of links at low utilizations and insensitive to propagation and queuing delays at high utilizations [20]. Therefore at high network utilizations, the value of the cost based on ‘hop-normalized-delay’ is always lower than the cost when ‘delay’ is used as definition. This is the basic idea behind ‘hop-normalized-delay’, which makes the links that were highly utilized during the last link update more attractive to the packets than other cost definitions, in order to get a better throughput and load balancing in the network [20]. The logic behind this idea is that the links with high utilization during the last link update do not necessarily remain highly utilized till the next link update and may still be used for sending packets.

It is interesting that also the relation between the throughput and the delay per packet has been changed. Although there is a waterfall in the throughput, the average delay per packet remains relatively constant in contrast to the case with no constraints.

Introducing the constraints has no effect on the time consumed by QOSPF. Flooding is done periodically and independent of the network state, so the amount of calculations (broadcasting, updating, routing) remains the same. The only change is that some calculations will result in no path.
7.2.2 *Threshold based update policy*

This update policy has more ‘intelligence’ than the periodic update policy, which leads to the expectation that certain performances of the network should improve by using this policy.

No Constraints

The threshold policy in figure 7.11 shows a higher and a more stable average throughput than the PB policy. The throughput decreases slightly at higher TB values, because a higher TB value means less updates and so less updated information at the routers. Because of the relation between the throughput and the delay per packet, the latter also has an approximately stable performance.

![Figure 7.11. The total network throughput of the three routing algorithms with respect to the TB value. The cost is defined as ‘hop-normalized-delay’.](image)

Also the amount of time consumed by QOSPF has decreased with approximately a factor 9 in comparison with the PB policy. The factor 9 is calculated by taking the average of the time consumed by the three algorithms for the case of PB and dividing it by the same value for the case of TB policy. The decrease in the time is caused by the more intelligent way of triggering the link state update. The routers update their routing tables whenever a considerable change has occurred on their links. In this way a rapid change of the throughput that results in a waterfall effect for PB update, is missing when TB update is used.

Figure 7.11 shows a relative independency of Samcra for the value of the TB policy. When the TB value has become large enough, the lowest cost algorithm decreases in throughput and also in its delay per packet. This behaviour can be studied further by looking at the load balancing of the network. Figures 7.12 and 7.13 show the utilization of all the 45 links of the network for the Samcra hbh and lowest cost algorithm with a TB value of 80%.
Figure 7.12. The utilization of the links for lowest cost algorithm and TB value of 80%

Figure 7.13. The utilization of the links for Samcra hbh and TB value of 80%

The figures show that Samcra hbh has a better load balancing as there are more links utilized than with lowest cost algorithm. This load balancing results in more lightly utilized links, which in turn results in lower delay per packet.

Hold timer
By introducing a hold timer, the performance of the TB policy shows a strong resemblance to the PB policy (figure 7.14). By choosing relatively small threshold values (< ± 20%), the effect of the hold timer will dominate. If the threshold value is small enough, the time between broadcasting triggered by the TB policy will be shorter than the hold timer and so the hold timer will dominate and introduce a PB-policy-like performance. By choosing large values of threshold, and so making the triggering period larger than the hold timer, the threshold policy will dominate.

Figure 7.14. The total network throughput with a TB value of 20% and hold timer.

Constraints (9-4)

Figure 7.15. Total Network Throughput with TB policy and constraints (9,4). Cost is defined as ‘hop-normalized-delay’.
Figure 7.16. Average delay per packet with TB policy and constraints (9,4). Cost is defined as ‘hop-normalized-delay’

From the above figures it can be seen that the delay as well as the throughput have decreased. The results show a larger throughput value when ‘utilization’ or ‘hop-normalized-delay’ are defined as the cost than the case that ‘delay’ represents the cost. This was also the case when the PB policy was used. In figure 7.16, the delay remains under the value of 9 ms. By lowering the constraints the delay can even be less than 7 ms at the cost of the throughput.

Introducing the constraints (9,4) results in approximately 2 times less consumed time in the QOSPF component. In the case of TB, flooding is dependent on the state of the network. Introducing constraints reduces the throughput, the number of utilized links and causes less fluctuations in the link states. In this way, flooding is triggered less often, which results in less time consumed at the routers.

An interesting point is that when constraints have a finite value, the average value of the delay per packet of Samcra ER always exceeds the packet delay of Samcra hhb even with PB update policy. As already explained above, the effect of availability of very updated information at the routers dominates the advantages of calculating the paths source-based. In this case, where constraints are introduced, the network is very lightly loaded, which results in relatively few changes in the link states. For this reason, the information at the routers is relatively well updated in which case Samcra hhb performs better than Samcra ER.
Setting up a hold timer results again in a waterfall behaviour (figure 7.17). The simulation results that had pretty constant values, now show a drop at higher values of hold timer. The delay per packet remains the same and does not exceed the constraints. It can be noted that the average throughput of Samcra ER remains lower than that of Samcra hbh. The explanation for this may also be found in the better behaviour of Samcra hbh at lightly loaded networks, where the routers have well-updated information about the network.

![Figure 7.17. Total Network Throughput with TB policy and constraints (9,4). Cost is defined as ‘hop-normalized-delay’](image1)

![Figure 7.18. Average delay per packet with TB policy and constraints (9,4). Cost is defined as ‘hop-normalized-delay’](image2)

7.2.3 Comparison between PB and TB policy
TB policy has a more constant behaviour than PB, which results in a higher average throughput. Because of a more intelligent way of updating the links, the time consumed by the routers is much less.

Introducing finite constraints reduces the throughput, but it is noticeable that the maximum values of the throughput in case of TB and PB are the same for the three cost definitions. Defining the cost as ‘delay’ results in less throughput than the other two definitions.

Introducing constraints seems to filter the oscillations in the throughput. In case of PB policy there is still a waterfall effect but with flat edges. The TB policy has no waterfall effect and in the case that constraints are set, the throughput will be constant and independent of the TB value.

### 7.2.4 General conclusions

**Update policy:**
In the simulated environment, the threshold policy has proved to perform better than PB policy:

- Threshold policy does not have a waterfall effect, as it is the case with PB policy. This means less dependency on the value of the update period, which results in a higher average throughput and a more stable performance. A more stable performance may be useful when there is a change in the update values of the network. This change could be set automatically by the routers or manually by the administrator during the network operation, to optimise the performance of the network.

- Using threshold policy reduces the cost of flooding dramatically at the routers in comparison to period based update policy. This means less cost at the routers in terms of computation time for recalculating the routing tables and less network overhead in terms of packets used for flooding.

- The two update policies produce the same maximum value of throughput and average delay per packet on introducing constraints for Samcra. However threshold based update has again a very constant behaviour for different values of TB in contrast to PB, which again has a waterfall effect for large values of update period.

**Routing algorithm:**

- In case of PB policy a trade off must be made between the computation cost at the routers and the network throughput. If the goal is to achieve high throughput, Samcra hhb and lowest cost algorithm may be used in combination with short update periods. The delay per packet in both cases is the same. Short update periods result in high computational cost at the routers. But if the goal is to lower the computational cost at the routers, Samcra ER
may be used in combination with long update periods, where the throughput will be higher than the other two routing algorithms.

- In case of TB policy, Samcra hhb performs the best in terms of throughput over the whole range of the simulated TB values (10% - 100%). The delay per packet is then almost the same as that of Samcra ER, which is slightly higher than that of the lowest cost algorithm.

- Between the simulated routing algorithms, Samcra ER is the most constant one in terms of throughput and delay. This property can be used for a good forecasting of the network behaviour, especially when the update values change dynamically during the operation of the network.

- With introduction of constraints, the delay per packet can reasonably be controlled at the cost of less throughput. When the cost is defined as ‘delay’, the delay can even be forced to a lower value than the delay of the lowest cost, but the throughput will decrease depending on the relative strictness of the constraints.

**Cost definition:**

- For both policies, the three cost definitions perform the same, but by introducing constraints for Samcra, the ‘utilization’ and hop-normalized-delay as cost definition, perform much better than ‘delay’ in terms of the amount of throughput and the relative independency of the update value.

### 7.2.5 Discussion

As mentioned in paragraph 3.1.1, IntServ provides three services. Each of these services will be discussed here in the light of the above conclusions and recommendations will be given for choosing the right update policy and routing algorithm to deliver each of these services in our simulated network.

**Guaranteed service**

**Requirements:** controlled bandwidth and delay.

**Recommendation:**

The cost may be defined as either ‘utilization’ or ‘hop-normalized-delay’, because of their high throughput when constraints are introduced. Threshold policy may be chosen, because of its higher throughput at large update periods. This can reduce the computational costs at the routers. As routing algorithm, Samcra hhb or Samcra ER may be chosen because of the lack of the delay control with lowest cost algorithm. However Samcra ER is recommended, because of its ability to optimise an end-to-end path. As Samcra ER performs well with large update periods, this combination has been recommended to lower the computational cost at the routers.
**Controlled-load service**
Requirements: controlled bandwidth
Recommendation:
   The cost may be defined as either ‘utilization’ or ‘hop-normalized-delay’, because of their high throughput when constraints are introduced.
Threshold policy may be chosen, because of its higher throughput at large update periods. This can reduce the computational costs at the routers. All three routing algorithms may be used, but it is important to take into account the update value at which the performance is the highest. Delivering the desired service is then mostly the task of the packet schedulers.

**Best-effort:**
   As this service is not a QOS-aware request, we recommend to make the choice of parameters based on other considerations than the network behaviour. These considerations could be the cost of implementing a certain update policy or routing algorithm or the existence of a combination of several parameters in the same network used for guaranteed or controlled-load service.
8. Conclusions

This chapter consists of two parts. The first part discusses the conclusions mainly regarding the simulation results and the second part presents the conclusions regarding the simulator used in this thesis.

8.1 Conclusions regarding the simulation results

Delivering QoS has become one of the most important topics of research on the Internet. Much research is being done on developing new and improved Internet architectures to improve the performance of the current Internet. As these architectures may be implemented on a very large scale, they must have a reduced complexity and a high degree of compatibility with other existing networks.

Several important Internet architectures have been treated in this thesis, namely:
- Integrated Services architecture
- Differentiated Services architecture
- MPLS

The structures of these architectures have been discussed at the level of their components.

In the mentioned architectures, there are many components for which the input and the output are defined, but with the internal structures still left open for research. One of these components is the routing component that is responsible for calculating the paths for packets and making the routing tables at the routers. Although one intuitively can feel the importance of the routing component in a network, not much research has been done to implement more QoS aware routing algorithms in the above-mentioned Internet architectures.

The main goal of this thesis was to study the network performance when using Samcra, a QoS routing algorithm. Therefore, Samcra has been implemented in QRS1.1, which is an Integrated Services simulator. The simulation results form the basis of the conclusions presented below.

We have simulated three QoS routing algorithms:
- Lowest cost algorithm
- Samcra hop-by-hop (hbh)
- Samcra explicit routing (ER)

Furthermore, the effects of the routing algorithms have been studied in combination with two different link state update policies:
- Periodic update policy
- Threshold update policy

Based on our simulations we conclude the following:
• Higher throughput generally results in higher utilized links and buffers, which in turn result in a higher average delay per packet.

• For smaller values of the update period, the throughput is generally higher than at larger update periods. This is because of the more updated information available at the routers. The opposite of this behaviour is true for the average delay per packet as mentioned in the previous point.

• The threshold update policy is superior to the periodic update policy in terms of throughput and computational cost at the routers especially when the values of the update policies become larger.

• For a further decrease of the computational cost at the routers, a hold timer may be introduced. A relatively large hold timer will dominate the behaviour of the update policy and introduces a network behaviour similar to the case when period update policy is used with an update period equal to the hold timer value. Relative small hold timers will have little to no effect on the network as they may be smaller than the update periods of the update policy itself.

• In our simulations, the general behaviour of the network deteriorated after introducing a hold timer. This is mainly because of the small number of flows.

• Samcra can calculate a path based on two or more constraints. In this way, more optimised paths can be calculated than the lowest cost algorithm, which can only calculate a path based on one metric. Defining constraints provides the option to control the end-to-end values of the metrics along the path. These constraints may be set by the user, the network administrator or automatically by the network itself. This property is extremely useful for improving the QoS and is not available in the lowest cost algorithm. The metrics may be defined as ‘delay’, ‘number of hops’, etc.

• Introducing constraints puts certain limitations on the calculated paths. This will result in less qualified paths, which in turn results in less throughput. By defining one of the metrics as ‘delay’, it was even possible to force the average delay per packet to an even lower value than that of the lowest cost algorithm, but at the expense of a very low throughput.

• The general performance of Samcra exceeds that of the lowest cost algorithm. When no constraints have been defined, the throughput of Samcra is almost always larger than that of the lowest cost, while its average delay is only slightly higher.

For delivering QoS to real-time flows, a choice must be made between Samcra hbh and Samcra ER. We recommend the use of Samcra ER. Although, the limitations of our simulator did not allow us to have a better understanding of the difference in behaviour between these two algorithms, the following observed characteristics have led to our preference for Samcra ER:

• A hop-by-hop calculation of the path may in some cases result in less optimal paths or in the worst case even in a non-optimal path. The possibility of the occurrence of such behaviour is reduced when using explicit routing.
• Samcra ER has a more stable performance in terms of throughput and average delay per packet with respect to variations of the update period.
• Samcra ER has in general a higher throughput than Samcra hbh when less updated information is available at the routers. This can reduce the computational cost at the routers and at the same time improve the performance of the network in terms of throughput and average delay per packet.

8.2 Conclusions regarding the simulator

QoS Routing Simulator version 1.1 (QRS 1.1) [17] is used for the simulations. QRS is based on the well-known simulator Mars [18]. Although Mars has a graphical interface to define and monitor network components, QRS1.1 is a text-based simulator where the network configuration as well as the output are in text format. Furthermore it is a packet-level simulator, which indicates that the simulator monitors the network performance at the level of packets.

8.2.1 Advantages of QRS1.1

• The simulator provides a very wide range of parameters that can be set for each component.
• QRS1.1 is free-to-download. In this way it can be used for simulations without any copyright limitations.
• QRS1.1 is an open-source program, which allows the user to implement new components.

8.2.2 Disadvantages of QRS1.1

• At the start of using QRS1.1, the first problem that the user will face is the lack of a detailed manual. The existing manuals give a first impression of the program and let the user find out how the program works by trial and error.
• The few comments in the c-code of the program and the lack of a detailed description of the design of it make it difficult for the user to implement additional code. During this thesis an attempt has been made to solve this problem and the previous one, by writing a more detailed manual about some elements of the program that have been used during the thesis.
• As the network configuration takes place in a text file, it is very complex and time consuming to define a simple network. Moreover, the large size of the configuration file makes defining and modifying it extremely error sensitive. Table 8.1 gives some statistics about the size of the configuration file of the network used for the simulations in this thesis, consisting of 219 components:

<table>
<thead>
<tr>
<th>Lines</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

58
Components | 4800 | 80  
Interconnecting components | 800 | 12  
Total | 5600 | 92

Table 8.1. The size of the configuration file in lines and pages.

- The two main limitations of QRS1.1 are
  1- the maximum number of nodes = 50
  2- the maximum number of real time source = 30
This is not enough for studying the behaviour of routing algorithms or flow dynamics in a network. Even if the program allows more than 50 nodes, the complexity of the configuration file will make it extremely difficult for the user to define a network with 50 or more nodes.
Although a source may generate many flows during a simulation, the flows are all identical and have the same destination.

- QRS1.1 allows the user to monitor the value of many network parameters in respect to time. The output, which may contain three large columns, may be seen after the simulation. The problem is that values of all the parameters are stored unsorted in one file. To monitor 5 different parameters, one should run 5 different simulations for each parameter or simulate once and write a code-file to sort the 5 output values from the output file.
9. Recommendations

Based on the gained experiences during this thesis, the following subjects are recommended for a future study:

- Implementing Samcra in a Differentiated Services architecture and studying its behaviour in combination with other network parameters.

- Implementing Samcra in a simulator with both the Integrated Services and Differentiated services architecture support, where many nodes and flows are permitted and:
  1. Make a comparison with the simulation results found in this thesis.
  2. Study the behaviour of Samcra in a DS domain and make a comparison with its behaviour in IntServ architecture.

- I recommend the use of the event-driven, flow-level, QoS simulator, which has recently been developed by Bojan Lekovic\(^1\), especially for the MSc students who are intending to do their thesis at this group\(^2\). There are two reasons for this:
  1. There are no limitation on the number of nodes and flows.
  2. Because of its higher abstraction level (flow level in stead of packet level) it is easier to understand the structure of the program and add new components.
  3. The presence of the author of the program at this working group.

Depending on the subject of the research, some additional c-code must probably be implemented.

Although I do not recommend using QRS, I mention that the version 2.0 of QRS has been released, which is called: Extended QRS. EQRS supports Differentiated services, but as far as I know, it still has a text based interface.

- Studying the effect on the network performance by using different routing algorithms in combination with equal class update policy, unequal class update policy and moving average [21].

---

\(^1\) Ir. Bojan Lekovic, PhD student at TU-Delft, faculty of electrical engineering, the Telecommunications and Traffic-Control Systems (TVS) group.

\(^2\) The group of Telecommunications and Traffic-Control Systems (TVS), faculty of electrical engineering, TU-Delft.
Appendix. Additional Manual of QRS
Quality of Service Routing Simulator Version 1.1

Introduction

QRS1.1 is the simulator I used during my thesis. In doing so, I gained some experience with the program. As the main part of my thesis, I had to modify the program in order to implement a routing algorithm called SAMCRA [Van Mieghem, P., H. De Neve and F. Kuipers, 2000, "Hop-by-hop Quality of Service Routing"]). From the beginning, my biggest problem was the poor manual of QRS and the very few comments written in the program. As QRS is based on MARS, reading the manual of MARS besides that of QRS1.1 is strongly recommended. However, not enough information could be found about the internal structure of the program or how to run simulations and view the desired output.

This is the reason why this manual has been made. This manual should give more detailed information about working with the program and also how the program itself works, of course only in the extend of my thesis-subject.

It is important not to regard this manual as a full replacement of the existing manual of QRS 1.1, but more as complementary to it. Therefore it is wise to first read the QRS manual before this one.

The first part of the manual is about how to work with the program, run simulations and get the desired output.

The second and the third part of the program give a deeper look into the program and describe how respectively the routing and link state update are implemented in the program.

I hope that this manual will make the work of the future thesis students, who will use QRS1.1, more easily, which is the main goal of this manual.
Part I

1. A general view

QRS 1.1 is a packet-level simulator written in C, which allows the user to study many QoS aspects in IP networks. The user can define an arbitrary network, run simulations on it and log the desired output values.

Where to download QRS 1.1 and how to install it, is already been explained in the original manual of QRS 1.1. This original manual as well as the simulator itself can be downloaded from http://www.tct.hut.fi/u/pgzhang/QRS/index.html.

2. Network configuration

QRS 1.1 has only a command line interface. This means that the input file as well as the output file is a text file. The input file has the extension *.cfg. An example of an input file can be found at the end of this manual.

An input file consists of several components, which are defined individually. Every component has its own parameters, which can be set by the user. How these components are connected, is defined at the end of such an input file. A graphical view of how the components should be attached to each other is shown in figure 1.

![Figure 1. An example of a network in QRS1.1](image)

Figure 1 shows a network with two nodes. Every block represents a component in the configuration file. Each node is attached to four components and one or more links. So if there are two nodes in the network, one should define at least 10 other components: four for each node, one link and a link cost component. There are also two separate components in the figure. The Link Cost component should only be defined once in the configuration file and is not attached to any other component.

The second component, which is called the Performance Monitor, is not necessary for a good operation of the network, but as it’s name suggests, it is only a monitor component which will log several pre-defined values about the network, which can be viewed after a simulation.

QRS provides the next components:

- NODE
- LINK
- RM
RSVP
QOSPF
REALTIME_SOURCE
REALTIME_SINK
SIMPLE_SOURCE
SIMPLE_SINK
TELNET_SOURCE
TELNET_SINK
FTP_SOURCE
FTP_SINK
LINK_COST_FUNC
PERF_MON
STOPPER

Chapter 2.1 will give a general description of an input file, and explains how different components and parameters can be set and/or logged. Chapter 2.2 will give a more detailed view of every component and explains the meaning of most of the parameters to the extend of the experiences gained during this thesis. Finally chapter 3 discusses the c implementation of several routing algorithms in QRS1.1.

2.1 A general overview of the input/output file

As it can be seen in the input file example at the end of this manual, there are several components in a configuration file. Each component consists of two groups of parameters named ‘param’ and ‘pflags’. param is always an input parameter and pflags is always an output parameter.

A line with ‘param’ can be something like this:

\[\text{param 0 82 0} \quad \# \text{Routing method: 0-> lowest cost, 1-> wid, 2->SAMCRA}\]

- \(\text{param}\) indicates that the following is an input parameter, which can be set by the user.
- The first 0 is the value, which can be set. In this case if ‘0’ is chosen, the lowest cost routing algorithm will be used, ‘1’ means that the routing algorithm the widest bandwidth will be used and so on.
- 82 0 is some standard value and do not have to be changed.
- \# means that the program will not read the text after this sign.

A line with a ‘pflags’ can be like this:

\[\text{pflags 2e 4} \quad \# \text{Packets Sent: 950}\]

Explanation:
- \(\text{pflags}\) indicates that this is an output parameter.
- 2e means that the input is not logged into a log file. By changing this value into 6e the value of this parameter will be logged into a file named sim_log.ID, which can be viewed when the simulation is finished. ID is just a random number assigned to the file. If there are no pflags with the value 6e, than no sim_log.ID file will be produced. Assume that the pflags value of both parameters ‘packet sent’ and
‘packets received’ of a source component are set to 6e. The sim_log.ID file can then look like this:

```
# simulation started with seed 956757654 on ws2
# 1 ‘realtime_source_1’ ‘packets sent’
# 2 ‘realtime_source_1’ ‘packets received’
1 0 0
2 0 0
1 120 1
2 120 0
1 140 2
1 158 3
....
```

There are 3 columns. The first column indicates which parameter is logged according to the explanation on the second and third line. So if the value of the first column is ‘1’ it means that that line gives the value of the ‘packet sent’ of the ‘realtime_source_1’.

The second column represents the time on which the value is measured.

The third column gives the value of that parameter. So at the time 120, the source had sent its first packet and at the time 140 its second packet and so on. We can also see that the value of ‘packets received’ remains ‘0’, which is of course a very logic behaviour of a source!

- After the sign ‘#’ we can see the value of Packet sent = 950. The explanation is that after running a simulation, beside a sim_log.ID file, another file will always be produced named sim_snap.ID. This file is identical to the original input file with the only difference that the values of ‘pflags’ after the ‘#’ sign are all updated.

Therefore the line:

```
pflags 2e 4      # Packets Sent: 950
```

in an input file has no real particular meaning. But after running a simulation, these lines will all be set into the sim_snap.ID file with their values updated. So then this line in an output file means that, the corresponding source has sent 950 packets during the whole simulation.

It is important to mention that in a configuration file, the parameters of each type of component must be configured in order, if the user changes the order of any two parameters, it might lead to unexpected results. Therefore, it is strongly suggested that the user configures the network configuration file through copying and modifying the example configuration file. Moreover, the user must not change ‘pflags’ to ‘param’ or vice versa.

### 2.2 A more detailed view of the components

In this paragraph, the content of each component will be printed and discussed. The numbers between parentheses after the lines, are merely added for an easy reference to these lines and are not part of the file.
The components discussed here, can all be found in figure 1.

2.2.1 Node

A node component models the “physical” aspects of a store-and-forward entity. QRS 1.1 supports a maximum of 50 nodes. Each NODE component must be connected with one RSVP, one RM, one QOSPF and at least one LINK component. The definition of this component in the configuration file is as follows:

```plaintext
component 'node0' NODE 0 0  (1)
param 'node0' 32 0   # node0  (2)
param 1000 82 0     # Delay to process a packet (µsec): 1000  (3)
pflags 0 0          # Speed of node (µsec/Kbytes): 0  (4)
pflag -1 82 0       # Buffer space in bytes (-1=inf): -1  (4)
pflags -1 82 0      # Mean time btw failures (sec): -1  (5)
pparam 1 82 0       # Interfailure dist (0=>EXP, 1=>UNIF): 1(6)
pparam 0 82 0       # Enter standard deviation if UNIF: 0
pparam 1200 82 0    # Mean time to repair (sec): 1200  (7)
pparam 0 82 0       # Repair time dist (0=>EXP, 1=>UNIF): 0
pparam 0 82 0       # Enter standard deviation if UNIF: 0
pflags 26 0         # Node status: Up
pflags 2e 4         # Buffer space used: 137744
pflags 2e 4         # Max buffer space used: 137908
pflags 2e 4         # Number of packets dropped: 0
pflags 2e 4         # Instantaneous drop rate: 0
pflags 2e 4         # Memory utilization: 137744
pflags 2e 4         # Input routing queue has 2633 pkts
pflags 2a 8         # flow table
pflags 2e 4         # 1k0-1 output queue has 0 pkts
pflags 2e 4         # 1k0-4 output queue has 0 pkts
```

1- Component indicates that this is the beginning of a new component. 'NODE’ indicates the type of the component.

2- ‘node0’ is the name of the component, which is specified by the user. Any component must have a unique name. After linking QRS 1.1 to SAMCRA, it has become necessary that the name of the nodes must have the form ‘nodex’ with x an integer starting from ‘0’. Note that the name ‘node0’ will be regarded the same as ‘node00’ and ‘node01’ will be treated the same as ‘node1’, so avoid this kind of naming!

3- This parameter sets the delay of a node in µsec.

4- This parameter indicates the speed of the node in µsec/Kbytes.

5- This is the mean time of failure of the node. By setting this value into –1 the node will never fail.

6- This will give a certain distribution to the time to failure. So the time that is filled in as the mean time between failures will be the mean of that distribution. The standard deviation of the distribution can also be chosen

7- This is the time after which a node is again operational after a failure. By setting this value into –1 the node will never recover.
2.2.2 LINK

A link component models a transmission channel between two nodes. In QRS1.1, class based queuing (CBQ) is also implemented in Link component. Therefore, A Link component is characterized by parameters of link and parameters of CBQ. By connecting Link components and Node components, the user can specify a network of arbitrary topology.

The Link component must be connected to 2 nodes. The Link component has the following structure:

```
component 'lk26-27' LINK 0 0 (1)
param 'lk26-27' 12 0     # lk26-27
param 1000 82 0          # Link propagation delay (MSECS): 1000 (2)
param 1.5e+06 82 0       # Link bandwidth (bytes/sec): 1.5e+06 (2)
param -1 82 0            # Mean time btw failures (msecs): -1
param 1 82 0             # Interfailure dist (0=>EXP, 1=>UNIF): 1
param 0 82 0             # Enter standard deviation if UNIF: 0
param -1 82 0            # Mean time to repair (msecs): -1
param 0 82 0             # Repair time dist (0=>EXP, 1=>UNIF): 0
param 0 82 0             # Enter standard deviation if UNIF: 0
param 0.8 82 0           # CBQ: EWMA weight : 0.8 (3)
param 1.5e+06 82 0       # CBQ: EWMA average limit : 1.5e+06
param 50000 82 0         # CBQ1: class A queue size : 50000 (4)
param 50000 82 0         # CBQ1: class B queue size : 50000
param 50000 82 0         # CBQ1: class C queue size : 50000
param 50000 82 0         # CBQ1: class D queue size : 50000
param 250000 82 0        # CBQ1: Class A queue bandwidth : 250000 (5)
param 500000 82 0        # CBQ1: Class B queue bandwidth : 500000
param 500000 82 0        # CBQ1: Class C queue bandwidth : 500000
param 250000 82 0        # CBQ1: Class D queue bandwidth : 250000
param 50000 82 0         # CBQ2: class A queue size : 50000
param 50000 82 0         # CBQ2: class B queue size : 50000
param 50000 82 0         # CBQ2: class C queue size : 50000
param 250000 82 0        # CBQ2: class D queue bandwidth : 250000
param 50000 82 0         # CBQ2: class A queue bandwidth : 250000
param 500000 82 0        # CBQ2: class B queue bandwidth : 500000
param 250000 82 0        # CBQ2: class C queue bandwidth : 250000
param 250000 82 0        # CBQ2: Class D queue bandwidth : 250000
pflags 26 0              # Link status: Up (6)
pflags 6 0               # Failure status: 0 (7)
pflags 2e 2              # Inst. Data Util., node26->node27: 0.0186667
pflags 2e 2              # Inst. Rout. Util., node26->node27: 0.0231467 (8)
```

1- After linking QRS 1.1 to SAMCRA, it has become necessary for a Link to have a name with the structure ‘lkx-y’ where x and y are integers and are equal to the integers in the name of the nodes which are linked with the Link component. So the Link between ‘node5’ and ‘node17’ must be named: ‘lk5-17’ or ‘lk17-5’.

2- This is the bandwidth of the link in Bytes/s.

3- In the component of a LINK, the values of EWMA (Exponential Weighted Moving Average) should be carefully specified. The ‘EWMA weight’ must be
set within (0,1). The value of ‘EWMA average limit’ might be set to a relatively large value (for example 150000) when the user has no aim to study the effect of CBQ. In this case, CBQ becomes Priority queuing (PQ).

4- In QRS1.1 there are 4 types of classes:
5- Class A: is used for the signalling packets
6- Class B: is used for real-time flows and has a higher priority than Class C
7- Class C: is used for real-time flows and has a lower priority than Class B
8- Class D: is used for best effort flows like telnet, ftp and simple flows.
9- A Link is bi-directional. CBQ1 refers to one direction and CBQ2 to the other. The sum of the bandwidths of different classes must be equal to the total bandwidth available on that link.
10- This value gives the last measured value of the link utilization. By setting the value of pflags into 6e as mentioned before, it is possible to log the value of the link utilization during the whole simulation.
11- This value is the link utilization caused by routing packets, so not the real actual data.
12- This value is the same as point ‘5’ above, but in the opposite direction.

2.2.3 Resource Management, RM

RM has no input parameters.
RM component must be connected to one NODE, one RSVP and one QOSPF component.

```
component 'rm0' RM 0 0
param 'rm0' 12 0 # rm0
pflags 2a 8 # Local topology table
pflags 2a 8 # flow table
```

2.2.4 Resource Reservation Protocol, RSVP

RSVP sets up paths for forwarding data traffic of Real-time Traffic. RSVP performs path setup hop by hop.
The RSVP component must be connected to one NODE, one RM, QOSPF and a REALTIME SOURCE / SINK component.
RSVP has no input parameters.

```
component 'rsvp0' RSVP 0 0
param 'rsvp0' 12 0 # rsvp0
pflags 2a 0 # Current flows in use: 0 (1)
pflags 2a 0 # Average RSVP packets per time interval: 0 (2)
pflags 2a 0 # Total RSVP packets: 1 (3)
pflags 2a 0 # Total RSVP PATH packets: 0 (4)
pflags 2a 0 # Total RSVP RESV packets: 0 (5)
pflags 2a 0 # Total RSVP PATH_ERR packets: 0 (6)
pflags 2a 0 # Total RSVP RESV_ERR packets: 0 (7)
pflags 2a 0 # Total RSVP PATH_TEAR packets: 1 (8)
pflags 2a 0 # Total RSVP RESV_TEAR packets: 0 (9)
pflags 2a 0 # Average RSVP packets per time interval: 1
```
The output parameters are
1- Number of flows [bytes/sec]
   The parameter records the number of active flows in the flow table of RSVP at
   node.
2- Average RSVP packets
3- Total number of RSVP packets
4- Total number of PATH packets
5- Total number of RESV packets
6- Total number of PATH_ERR packets
7- Total number of RESV_ERR packets
8- Total number of PATH_TEAR packets
9- Total number of PATH_TEAR packets

2.2.5 Quality of service OSPF, QOSPF

QOSPF is responsible for path calculations and link state update. QOSPF must be connected to one NODE, one RSVP and one RM component.

```
component 'qospf0' QOSPF 0 0
param 'qospf0' 12 0 # qospf0
param 100000 82 0 # Time btw topology broad (msec): 100000 (1)
param 10 82 0 # Standard deviation: 10
param 0 82 2 # Routing method: 0->LC, 1->WB, 2->SAMCRA: 0 (2)
param 0 82 0 # LS update: 0->PB, 1->TB, 2->ECB, 3->UCB: 0 (3)
param 1000 82 0 # value of LS update period : 1000 (4)
param 0.1 82 0 # update by bandwidth percent variation: 0.1 (5)
param 0 82 0 # value of hold timer, used with other upd: 0 (6)
param 20 82 0 # the number of classes for equal class up: 20 (7)
param 0.01 82 0 # the size of base class B for unequal cla: 0.01 (8)
param 2 82 0 # for unequal class update, f>1 : 2 (9)
pflags 2e 8 # total time cost in QOSPF: 1.24975e+06 (10)
pflags 4 4 # Sequence number: 1001
pflags 2e 8 # Global topology table of Cost
pflags 2e 8 # Flow Global topology table of bandwidth
pflags 2e 8 # Local topology table
pflags 2e 8 # Routing table
pflags 2e 8 # Flow Routing table
pflags 4 0 # Last sequence no table
pflags 2e 8 # Flow table
```

1- This parameter is ignored in the case that the period based (PB) link state update method is used. In other cases, this parameter is used in case of deadlock. It should be set to a relatively large value, e.g. 10000. The unit is in msec.
2- This parameter indicates which routing algorithm will be used:
   Lowest Cost algorithm
   Widest Bandwidth algorithm
   Samcra hop by hop routing algorithm
   Samcra source-based routing algorithm
3- This parameter indicates which link state update policy will be used:
   Period Based
Threshold Based
Equal Class Based
Unequal Class Based

4- This is the value of the Period based update.
5- This is the value of the Threshold based update
6- This parameter specifies the interval of LS advertising in case that the frequency of LS changing is too high. The unit is 10 µsec. LS advertising is triggered only when the difference between the current time and last LS advertising time is more than the hold time.
7- This is the number of classes in the case of Equal Class Based
8- This parameter specifies the size of base class (B) if unequal class based (UCB) update method is used. B is in percent of the total link bandwidth.
9- This parameter specifies the value of factor (f) which is used to define unequal size classes: \[ 0 , B ] \[ B , (f+1)B \] \[ (f+1)B , (f^2+f+1)B \] … if Unequal Class Based (UCB) update method is used.
10- This parameter records the time consumed by the QOSPF in a node.

2.2.6 Real-time Source

There are several kind of sources supported in QRS 1.1, but since our project only focussed on the real-time sources / sinks, we will only refer to them in this manual. In QRS 1.1 each source has its own sink. The real-time source / sink pairs recognize each other by their value of ‘flow-index’, which should be the same for the source and the sink that are forming the pair. Other kinds of sources do not have such a ‘flow-index, but they also form pairs, which must be defined in the configuration file. In this way one source can only send to one sink. One should notice here that this does not mean that a node can only send to one other node. By attaching several sources to a node, that node can send many flows to different sinks.

QRS 1.1 permits a maximum of 30 real-time pairs, so in other words a maximum of 30 flows. A real-time source/sink should be connected to one NODE and one RSVP component.

```
component 'realtime_node0-node15_so' REALTIME_SOURCE 0 0
param 'realtime_node0-node15_so'12 0 # realtime_node0-node15_so
pflags 6 0 # Peer: realtime_node0-node15_so
param 0 82 0 # Not select(0), select(1) : 0
param 512 82 0 # Ave Packet length: 512 (1)
param 2000 82 0 # the time of starting to request RSVP aft:
 2000 (2)
param 1 82 0 # Choose dist (0=>EXP, 1=>UNIF, 2=>CBR, 3=: 1
param 0 82 0 # Enter standard deviation if UNIF or FBN: 0
param 5000000 82 # Enter the interval(us) of traffic produc:
 5000000 (3)
param 1000000 82 # Enter the interval(us) of traffic pause:
 1000000 (4)
param 1000 82 0 # Ave delay btw packets (µsec): 1000 (5)
param 0 82 0 # Choose dist (0=>EXP, 1=>UNIF, 2=>CBR, 3=: 0
param 0 82 0 # Enter standard deviation if UNIF or FBN: 0
param -1 82 4 # Window size (-1 for inf): -1
pflags 2e 4 # Packets Sent: 0 (6)
pflags 2e 4 # Packets Acked: 0 (7)
```
1- This is the average length of a packet in bytes.
2- This parameter specifies the time between the start of the simulation and the time that real-time traffic begins to request for path setup. The unit is 10 μsec.
3- This parameter specifies the period of time that real-time traffic is produced. The unit is in μsec.
4- This parameter specifies the period of time that real-time traffic is paused. The unit is in μsec.
5- This indicates the average delay per packet. The unit is in μsec.
6- This indicates the number of packets sent.
7- This is the number of packets acknowledged by the sink. This is an output parameter.
8- This indicates the number of packets received.
9- This is the number packets lost. This is an output parameter.
10- This is the flow-index of this source and as already mentioned, the sink with the same flow-index will be known as the pair of this source. So different pairs of real-time traffic must have different flow-index numbers.
11- The user can choose between 1 and 2. 1 indicates class B and 2 indicates class C.
12- This is the flow rate of this source. It is very important to take into account the following: In the c-code of QRS 1.1, this value is seen as the flow-rate of this source, but in reality the value of the flow rate is determined by two other parameters: the Average packet length and the Average delay between packets. To set the desired flow rate, one should calculate the correct value based on these two parameters manually and it is also very important to set the value of the flow rate of this source, which must also be done manually. Example: if the Average Packet Length is set to 512KB and the Average Delay between packets is set to 1000 μsec, then the real flow rate of the source will be: μsec / 1.000 μsec) * 512 KB = 512 KB/s. The value of the Flow rate of this source must then be set manually to 512.000.

2.2.7 Real-time Sink

A real-time source/sink should be connected to one NODE and one RSVP component. The definition of a real-time sink Component is as follows:

component 'realtime_node0-node15_si' REALTIME_SINK 0 0
param 'realtime_node0-node15_si' 12 0 # realtime_node0-node15_si
pflags 6 0 # Peer: node0.realtime_node0-node15_so
param 0 82 0 # Not select(0), select(1) : 0
param 512 512 82  # Ave Packet length: 512
param 2000 82 0  # Ave delay btw packets (µsec): 2000
param 0 82 0  # Choose dist (0=>EXP, 1=>UNIF, 2=>CBR, 3=: 0
param 0 82 0  # Enter standard deviation if UNIF or FBN: 0
param 20 82 4  # Window size (-1 for inf): 20
pflags 2e 4  # Packets Sent: 0
pflags 2e 4  # Packets Acked: 0
pflags 2e 4  # Packets Received: 0
pflags 2e 4  # Ack Packets Lost: 0
param 0 82 0  # Enter Hurst parameter (Not used): 0
param 1 82 0  # Flow index of this sink : 1
param 2 82 0  # Class type of this sink : 2
param 512000 86 0  # Flow rate of this sink : 512000 (1)
param 0 86 0  # Flow delay of this sink : 0
param 20000 86 0  # Refresh Period of this sink (10us): 20000
pflags 2a 8  # Flow table

This configuration of this Component is very much like a Real-time Source so not much additional information is necessary.
1- This value should also be set for the Sink and must of course be the same as its Source-pair

2.2.8 Link Cost Component

The Link Cost Component is defined only once in the configuration file and is not attached to any other component.

cOMPONENT 'CostFcn' LINK_COST_FUNC 0 0
pflags 2 0  # CostFcn
param 2 82 0  # Cost fcn 1:hndl 2:dly 3:util 4:hop: 2 (1)
param 100000 82 0  # Delay cost fcn max delay: 100000
param 10 82 0  # Delay cost fcn min delay: 10
param 10 82 0  # Slope: 10
param 0 82 0  # Offset: 0
param 2 82 0  # Cost movement limit: 2
param 10 82 0  # Maximum Cost: 10
param 1 82 0  # Minimum Cost: 1
param 0.5 82 0  # Exp filter, coeff of new: 0.5

1- With this parameter one can choose between several definitions of cost during a simulation cycle. So by choosing 4, the simulator will regard the number of hops as the cost and the algorithm of Lowest Cost tries to minimize this cost.
1- Hop-normalized delay [Khan and Zincky, 1989]: utilization is calculated as follows: The average queuing delay per packet and the average transmission delay per packet are measured over the last link-cost update period. The utilization is calculated from these delays, assuming an M /M/ 1 queuing model.
2- Delay: Represents the average delay suffered by a packet on the outgoing link queue and in transmission (including the processing time at the node and the propagation delay of the link), over all packets transmitted in the last link-cost update period
3- Utilization: Fraction of the time that the outgoing link queue (including the packet being transmitted) was not empty in the last link_cost update period.
4- Hop-count: 1 for a neighbour node, \( \infty \) otherwise.

For more detail about the rest of this component, we refer to the manual of MARS, version 1.0 [June 1, 1991].

### 2.2.9 Performance Monitor

The performance monitor component collects and evaluates statistics about the network. Average and instantaneous values of performance measures are calculated. An average value is computed based on statistics collected after a specified period of time, called the start-up interval. An instantaneous value is computed based only on statistics collected during the last update period.

A Perf Monitor component is not attached to any other component.

```
component 'Monitor' PERF_MON 0 0
pflags 2 0    # Monitor
pflags 2e 4   # Total network throughput: 9099.31 (1)
pflags 2e 4   # Inst network acked rate: 9646.08 (2)
pflags 2e 4   # Time var of inst acked rate: 1066.91
pflags 2e 4   # Average delay/packet: 11.1814 (3)
pflags 2e 4   # Inst ave delay/packet: 11.5458 (4)
pflags 2e 4   # Time var of inst delay: 0.543962
pflags 4 4    # Max delay/packet: 37780 (5)
pflags 2e 4   # Connection count: 65 (6)
pflags 2e 4   # Total selected net throughput: 0
pflags 2e 4   # Inst selected net acked rate: 0
pflags 2e 4   # Time var of inst sel. acked rate: 0
pflags 2e 4   # Selected avg delay/packet: 0
pflags 2e 4   # Inst sel. ave delay/packet: 0
pflags 2e 4   # Time var of sel. inst delay: NaN
pflags 4 4    # Max sel. delay/packet: 0
pflags 2e 4   # Selected connection count: 0
pflags 2e 4   # Selected packets Dropped: 0
pflags 2e 4   # FTP Avg Pkts/Conn: 0 (7)
pflags 2e 4   # FTP SD : Pkts/Conn: 0
pflags 2e 4   # FTP Avg Time of a Conn: 0 (8)
pflags 2e 4   # FTP SD : Time of a Conn: 0
pflags 2e 4   # TELNET Avg Pkts/Conn: 0 (9)
pflags 2e 4   # TELNET SD : Pkts/Conn: 0
pflags 2e 4   # TELNET Avg Time of a Conn: 0 (9)
pflags 2e 4   # TELNET SD : Time of a Conn: 0
pflags 2e 4   # Link failure count: 0 (10)
pflags 2e 4   # Packets Dropped: 0 (11)
pflags 2e 2   # Routing Packets in net: 528012 (12)
pflags 2e 2   # Data Packet Load: 0.325558 (13)
pflags 2e 2   # Inst. Data Packet Load: 0.347193 (14)
pflags 2e 4   # Time var. of Data Load: 0.0525207 (15)
pflags 2e 2   # Routing Packet Load: 0.0222042 (16)
pflags 2e 2   # Inst. Routing Packet Load: 0.0212812
pflags 2e 4   # Time var. of Routing Load: 0.000119394
pflags 2e 4   # Max occupied buffer space in net: 1099156 (17)
pflags 2e 4   # Min occupied buffer space: 108692 (17)
```
pflags 2e 4   # Ave occupied buffer space: 503104 (17)

1- The total number of data bytes that have been acknowledged during the measurement interval divided by the length of the measurement interval.
2- The total number of data bytes that have been acknowledged in the last update period divided by the length of the update period.
3- The delay of a data packet is the time difference between the first time the data packet is sent by the source into the network and the receipt of its corresponding ack packet at the source. The average delay per packet is the total delay of all data packets that have been acknowledged during the measurement interval divided by number of data packets acknowledged in this interval.
4- The total delay of all data packets that have been acknowledged in the last update period divided by number of data packets acknowledged in this period.
5- The maximum of the delays of data packets that have been acknowledged during the measurement interval.
6- Number of connections that are currently on.
7- The total number of data packets in all FTP flows that finished during the measurement interval divided by the number of such flows.
8- The active duration of an FTP train is the time interval from producing the first packet in the train to receiving ack packets for all packets of the train at the source. The average lifetime of an FTP connection is the total of the active durations of all the FTP trains that finished during the measurement interval divided by the number of such trains.
9- Defined similar to the FTP one.
10- Number of link components that currently has status Down.
11- Number of workload packets that are dropped during the simulation.
12- Number of routing packets used during the simulation.
13- The fraction of the network capacity (i.e. sum of all link capacities) used by workload packets during the measurement interval.
14- The fraction of the network capacity used by workload packets in the last update period.
15- The fraction of the network capacity used by routing packets during the measurement interval.
16- The fraction of the network capacity used by routing packets in the last update period.
17- At each node the maximum buffer space used is maintained. At the end of the simulation, the maximum, the minimum and the average of those are provided.
Part II

3. The structure of the program regarding Routing and Link state update policy

This chapter gives a detailed view of the C code of QRS1.1 regarding the routing and link state update policy (LSUp) in case of real-time traffic. This chapter is only useful for those users who are interested in adding certain features on the C-code of QRS1.1.
Both routing and LSUp are implemented in the file qrs/comps/qospf.c so the main focus of this chapter will be on this particular file.

The chapter is divided into three sections. In the first section a general overview will be given of the QRS implementation regarding real-time traffic. Section 3.2 discusses the implementation of the routing algorithms and finally 3.3 is about the LSUp.

3.1 A general overview of QRS1.1 regarding real-time traffic

Like any other simulator, QRS1.1 needs to have a network representation of the attached nodes, the value of different parameters of the links and so on. For a real-time traffic all these parameters are set in the file qospf.c.
As already mentioned, each node is attached to a QOSPF component. During a simulation, when a real-time flow enters a node qospf.c is called and run for this active node to fine the next hop requested by RSVP. So the values of different variables in qospf.c will not be the same during a simulation but rather change every time another node is considered and so adopt the values of that certain node.
The content and the properties of a QOSPF component can be seen in the file qrs/comps/include/qospf.h.
Every time the file qospf.c is called, all the properties set in qospf.h are assigned to the variable ‘g’, where ‘g’ is a struct.

3.1.2 Network representation

3.1.2.1 Global Topology Table of cost

One of the matrices that contain information about the network in QRS1.1 is called Global Topology Table of cost (GTT) with dimensions [number of nodes x number of nodes]. According to the chosen method for cost calculation in Link-Cost component, GTT matrix contains the cost between the nodes.
GTT is called as follows: GTT(g)[i][j]: this means the value of the i-th row and j-th column of GTT seen from node ‘g’. If nodes ‘i’ and ‘j’ are connected, the value of GTT(g)[i][j] will be the cost between those nodes otherwise the value will be INFINITY.
An important issue is that the content of GTT is not constant during a simulation, but rather changes! Row 1 always represents the node, which is at that moment active in the simulator. This means that if the file qospf.c is called for node15, row 1 of GTT will be setup for this particular node.
For more clarity, a gtt has been defined, which will remain the same throughout a simulation, when Samcra is chosen.

3.1.2.2 Flow Global Topology Table

The form of FGTT and also the way it is called are the same as GTT, but where GTT contains a ‘none INFINIT’ value, FGTT contains the value of the link capacities between those nodes. Also FGTT is not constant during a simulation! Its order of nodes is at all times, the same as that of the GTT matrix.

3.1.2.3 Local Topology Table

Local Topology Table (LTT) is a struct with very useful information about the network. The definition of LTT can be found in the file qrs/comps/include/route.h. As the name suggests, LTT gives local information about the node considered. This local information could be the links attached to the node, the nodes attached to the node, the cost of the links attached to the node etc. The way LTT is called is: LTT(g)[i].xxxxxx. ‘i’ is an integer with a value between ‘0’ and the Maximum-Number-of-Links-per-Node, which is 20. So when a node has two neighbours, ‘i’ will have the values 0 and 1. And if there are 20 neighbours ‘i’ will have the values (0, 19).

Some parameters:
- LTT(g)[0].n_nd : the first node attached to the current node. ‘n_nd’ is a struct!
- LTT(g)[0].n_nd->co_name : the name of the node LTT(g)[0].n_nd as set in the configuration file.
- LTT(g)[0].n_link : the link between the current node and LTT(g)[0].n_nd. ‘n-lnk’ is a struct!
- LTT(g)[0].n_link->co_name : the name of the link LTT(g)[0].n_link as set in the configuration file.
- LTT(g)[0].l_cost : the cost of link LTT(g)[0].n_link!

Figure 2 is a graphical expression of the above statements. Assuming that ‘node0’ is the current active node, the following is valid:
As there is no logical way to predict which node will be assigned to which integer, a function is provided in QRS1.1, which determines the index of LTT. This function is: `index_local_top_table (node, link)

So given that the node is node0 and the link is lk0-1, the function will return the value ‘0’, which refers to the integer between the brackets in LTT(g)[]. So LTT(g)[0].n_nd is ‘node1’

Note that in this function the input parameters ‘node’ and ‘link’ are of the type Component and have their own struct!

In case that the next node is given and the appropriate link must be found, the next function can be used:
`index_of (node, node)

So if the first node is node0 and the second node is node2, the function will return 1 and so LTT(g)[1].n_link is equal to the link component with the name ‘lk0-2’.

3.1.2.4 Flow Routing Table

Flow Routing Table (FRT) is a temporary variable, which is reset and set every time qospf.c is called. FRT is called by FRT(g)[i].xxxxxx. The integer ‘i’ behaves like the integer ‘i’ of LTT. FRT is a struct with 3 components:
FRT(g)[i].dest: it contains a destination node.
FRT(g)[i].hop: the link that leads to that destination node.
FRT(g)[i].cost: the cost of the path to that destination node.

These 3 components are reset at the beginning of routing a packet and then set again according to the routing algorithm used. LTT and the routing algorithm are used to set the values of FRT in each run and the values of FRT are in turn used for calculating the Flow Table, which determines the next hop of a particular flow. Flow Table will be discussed next.

The integer ‘i’ goes from 0 to ‘the number of nodes’. So if there are 17 nodes in a network, ‘i’ will range from 0 to 16.
So if we take the network of figure 2, again in the case that ‘node0’ is active, the FRT could be as follows:
FRT(g)[0].dest = node0 (node0 is a component and not a string!)
FRT(g)[0].hop = NULL (this is the link to ‘node0’ itself. That’s why the value is NULL)
FRT(g)[0].cost = 0 (this value of the costs are not shown in figure 2)

FRT(g)[1].dest = node1
FRT(g)[1].hop = lk0-1 (lk0-1 is a component and not a string!)
FRT(g)[1].cost = 3

FRT(g)[2].dest = node2
FRT(g)[2].hop = lk0-2
FRT(g)[2].cost = 3

FRT(g)[3].dest = node3
FRT(g)[3].hop = lk0-3
FRT(g)[3].cost = 2

In qospf.c there is also a function, which determines the right value of the integers of FRT called: index_of_FRT(node, link)
Its usage is in the same way as ‘index_local_top_table (node, link)’.
The function index_of(node, node) explained above can also be used here.
The struct of FRT is defined in the file qrs/comps/include/qospf.h.

3.1.2.5 Flow Table

Flow Table is a vector with as many rows as there are flows (the number of flows is equal to the number of real-time source / sink pairs). FT has also the form:
FT(g)[flow-index].xxxxx.
So if flow 3 passes a certain node, FT(g)[3].nhop determines the next hop of that particular flow. FT is a struct with many components, which can be found in the file qrs/comps/include/rsvp.h

3.2 Routing in QRS1.1

For handling the routing, the first thing of importance is to determine which routing algorithm is chosen by the user. ‘g->routing_method->u.i’ is the integer which is set in the configuration file, when choosing the desired routing method. According to this value, the right routing algorithm is chosen.

3.2.1 Lowest cost routing algorithm

This algorithm tries to find the path, which has the minimum cost and enough bandwidth. If there are more than 1 path with the same minimum cost, the first path that has been found, will be used.

The lowest cost algorithm operates as follows:
1- All the values of FRT(g)[i].cost are set to INFINITY and FRT(g)[i].hop is set to NULL. FRT(g)[i].cost represents here the cost of the link.
2- The value of the cost in FRT(g)[i].cost of the node itself is set to zero.
3- The cost of those links with a smaller bandwidth than the rate of the source, will be set to INFINITY. This also includes the links that have failed and are still not repaired. The rate is, as already mentioned, not the real rate of the source but the rate which has been set manually in the configuration file.
4- The link with the smallest cost is searched
5- If the costs are not smaller than INFINITY, terminate the program.
6- If a path exists to other nodes of the network, the value of the cost to these nodes is set to [the found minimum cost + the cost from the node with the minimum cost to the destination node according to GTT matrix].
7- The value of FT(g)[flow_index].nhop is then calculated according to FRT and is given to the simulator.

3.2.2 Widest bandwidth routing algorithm

This algorithm tries to find a path with the largest bandwidth available.

1- All the values of FRT(g)[i].cost are set to ‘0’ and FRT(g)[i].hop is set to NULL. The cost represents here the value of the link bandwidth.
2- The value of the cost in FRT(g)[i].cost of the node itself is set to INFINITY.
3- The cost of those links with a smaller bandwidth than the rate of the source, will be set to ‘0’.
4- The link with the largest cost (=bandwidth) is searched.
5- If the largest cost is equal to ‘0’, terminate the program.
6- If a path exists to other nodes of the network, the value of the cost to these nodes is set to Min[the founded minimum cost, the available bandwidth from the node with the maximum bandwidth to the destination node according to FGTT matrix].
7- The value of FT(g)[flow_index].nhop is then calculated according to FRT and is given to the simulator.

3.2.3 Samcra hop-by-hop

As the source code of Samcra had been provided by the authors [Van Mieghem, P., H. De Neve and F. Kuipers, 2000, "Hop-by-hop Quality of Service Routing"], the task was now to implement Samcra in QRS1.1. So the desired input of Samcra must be extracted from the information available at QRS1.1 and the output of Samcra must again be delivered to QRS1.1 in a manner understandable for the simulator. Samcra is implemented first in a hop-by-hop way, which means that every node decides what the next node should be. The second implementation is a source-based version of Samcra, where the path is only calculated by the source when generating a source. Other nodes will follow this source-based calculated path.
As Samcra uses adjacency lists for its network representation, the first step would be creating these matrices out of the available information in QRS1.1.

Part I
3.2.3.1 gtt matrix (global topology table)

For more convenience, a binary gtt (global topology table) matrix is defined with dimensions [50 x 50]. 50 is the maximum number of nodes allowed by the simulator. The matrix contains a ‘1’ if two nodes are connected and a ‘0’ otherwise. Note that this matrix has no connection with the previous defined GTT matrix.
gtt is defined and made in the file gtt_matrix.c, which is in turn called in qospf.c in the function static caddr_t qospf_start(g). In this function every node is scanned once and the name of its links are checked. According to the name of the links, the function gtt_matrix determines which nodes are connected to each other and in this way the gtt-matrix is set up. If the name of a link is ‘lk3-4’, then gtt[3][4] and gtt[4][3] are set to 1. That is why it is important to name the links (in the proper way) as mentioned, when describing the link component in §2.2.2.

3.2.3.2 Adding a new c-file

In order to add a new c-file to QRS1.1, the c-file could be put in the directory qrs/comps and the name of the file should be added in the Makefile that is in the same directory. In this Makefile there is a list of *.o files and *.c files. Add your file to both list with the name: ‘your-c-file.o’ and ‘your-c-file.c’.

3.2.3.3 Adjacency list

With the ‘gtt’ matrix, we can start with making three matrices, which will form the network representation of Samcra namely, the Numadj vector, the Adj matrix and the datadj matrix. The making of these matrices is done in the file adj_matrices.c.

Numadj is a vector with length [number of nodes]. Numadj[i] contains the number of nodes attached to node i.
The structure of the Adj-matrix is as follows: Row [i] contains as many integers as the value of the i-th column of Numadj. The integers in the row are the index of the nodes to which node ‘i’ is attached. So if Adj[i][1] = 3 and Adj[i][2] = 6, it means that the i-th node is attached to node 3 and 6.
The Datadj-matrix has 3 dimensions, where two dimensions are exactly the same as the Adj matrix and the third dimension contains additional information about the connection between the nodes like the delay or a cost. These additional parameters are called ‘metrics’.
So a Datadj with 2 metrics will have the form Datadj[x][i][j], where [x] can be ‘1’ or ‘2’ and [i] and [j] are the same as a Adj matrix. Note that the metrics must be additive! It means that they can be added together through the calculated path.
When the adjacency matrices are made, the memory occupied by gtt matrix is de-allocated.

Part II

Every time a real-time traffic flow comes to a node and qospf.c is called, the values of the link costs in the Datadj matrix are updated. In this way updated information of the link costs is always available especially after a link state update.
After this, the value of the costs of the links, whose last available bandwidths are less than the flow rate of the source, are set to INFINITY to prevent that link to appear in the calculated path.

Part III

The function of Samcra must be called as follows:

\[
\text{Samcrapath(source, destination, adj, numadj, datadj, number of metrics, the constraints, number of nodes, path, &nexthop);}
\]

- Source and destination must be integers.
- Adj, numadj and datadj have been already explained above.
- Number of metrics is an integer that indicates with how many metrics, Samcra should find the best path.
- The upper bound is the maximum value of the metrics that the calculated path should not exceed. If there are two metrics, the upper bound will be a vector with two values.
- Path and nexthop are the outputs of Samcra and have the following structure:
  - Path = [2 6 9 12 4 ]
  - Nexthop = 12
  - This means that the source is node 4 and the destination is node 2 and the path goes through the intermediate nodes 12, 9 and 6.

Now we have to prepare these input variables for Samcra.

3.2.3.4 Source

\text{g->node->co_name} is the name of the node, which is considered at that moment, and because of the hop-by-hop nature of the routing algorithm, this node automatically forms the source. However \text{g->node->co_name} is a string of the form ‘nodex’ and should be transformed into an integer. This is done by the function:

\[
\text{sscanf(g->node->co_name, "node%d", &node_concidered);}\]

\text{node_concidered} is an integer variable and with this function it gets the value of the integer ‘x’ of the string ‘nodex’.

QRS1.1 works with x-values starting from ‘0’ but Samcra starts from ‘1’, so the value ‘node_concidered + 1’ should be given to Samcra.

3.2.3.5 Destination

This is done in a similar way as preparing the source, but the used function is now:

\[
\text{sscanf(FT(g)[flow->index].dest_host->co_name,"node%d", &dest_node);}\]

The value “dest_node + 1” will be given to Samcra as Destination.

3.2.3.6 Adjacency lists

\text{Numadj, Adj and Datadj} have already been prepared and can be given to Samcra.
3.2.3.7 Number of metrics

Number of metrics used for calculating a path. When using ‘delay’ and ‘hops’, this value will be equal to 2. The values of these metrics is now stored in qrs/text_files/constraints. This solution is chosen, because adding this values within the c-code of QRS1.1, requires compiling the whole program when modifying one of these values. With this solution, compilation is not needed anymore.

3.2.3.8 Constraints

If there are 2 metrics, this will be a vector with size 2, [upperbound1, upperboud2]. If metric 1 represents the delay, then the total delay on the calculated path should not exceed the value of upperbound1 and if metric 2 represents the hop count, then the total hops in the calculated path should not exceed upperbound2. These constraints are now stored in a vector named ‘A’.

The result will then be:
\[
\text{samcrapath} \left( \text{node_concidered}+1, \text{dest_node}+1, \text{adj}, \text{numadj}, \text{temp_datadj}, 2, A, \text{number_of_nodes}, \text{path}, &\text{nexthop} \right);
\]

Part IV
This part concerns the preparation of the output of Samcra in order to push the flow to the nexthop calculated by Samcra.

First a check is performed to see whether Samcra has been able to find a path with the given restrictions of the upper bounds. If a path does not exist, Samcra return nexthop=0. In this case the simulator is given the command to go further and handle other flows.

Our nexthop is an integer. So the first thing that happens is a transformation of this integer to a string of the form ‘nodex’, where ‘x’ is equal to nexthop. This is done in the following way:
\[
\text{sprintf}(\text{next_node}, "\text{node}\%d", \text{nexthop}-1);
\]

If nexthop=10, then the string next_node will be ‘node9’. Nexthop-1 is used because of the same reason that (source +1) and (destination + 1) are used.

We have convert our integer in a string, but it is still not enough to give it back to QRS1.1, because the value of a node should be given to QRS1.1 as a component-type and not as a string. So now we have to convert our string to a component. The following c-lines, will scan the neighbour nodes to find a node with the same name as our next hop. Note that a node is of the type component, which is a struct, but its name is a string so we can compare it with our own string!

\[
\text{for} \ (i=0; \ LTT(g)[i].n_nd != (\text{Component \ *) \ NULL;} \ i++ \}
\text{if} \ (LTT(g)[i].n_nd \&\& \ (!\text{strcmp}(LTT(g)[i].n_nd->co_name,} \text{next_node})) \)
\]
With this code, the index of that node, which has the same name as our string, will be assigned to the value of LTT_ind. This means that LTT(g)[LTT_ind].n_nd is exactly the component we were looking for. Now is the transformation of an integer to a component completed.

As the value of the next hop is determined by FRT, the right values of the node should be set within FRT. We scan to see, which index of FRT is pointing to the destination node so we can change its value to the right next hop. For this we use the function \text{ind\_of\_FRT}:

\[
\text{ind} = \text{index\_of\_FRT}(g, FT(g)[\text{flow\_index}].\text{dest\_host});
\]

At the beginning of a simulation it can happen that this index cannot be found, because not every node has the total picture of the network. So a check is necessary to see if the value of \text{ind}=-1, which means that the destination node is unknown at that particular time of simulation. In this case we force the program to handle other nodes.

But if a valid ‘ind’ is found, the values of FRT(g)[ind] will be set equal to the values of LTT(g)[LTT_ind], which is already known.

\[
FRT(g)[\text{ind}].\text{hop} = LTT(g)[LTT\_ind].n\_link;
FRT(g)[\text{ind}].\text{cost} = LTT(g)[LTT\_ind].l\_cost;
\]

After these steps the memory used for the path matrix and the ‘temp_datalj’ is de-allocated, which is a temporary storage matrix for the values of Datalj matrix. On the next call of the qospf.c, these two can be calculated again.

3.2.4 Samcra source-based

In this part of the program, Samcra is implemented in a source-based way, which means that the path is calculated once by the source and the intermediate nodes pass the packet to the next node according to this source-calculated path. Parts I, II and III remain the same for the source-based version of Samcra.

Part IV

A matrix is introduced with dimensions [30 x number of nodes] and with the name ‘path\_matrix’ for storing the calculated paths by the sources.

So directly after Samcra has calculated a path, it is checked whether the current node is equal to the source node of the current flow. If so, it means that a new flow is being generated and the flow-index-th row of the path_matrix is reset and the function path_matrix_make is called to fill in that particular row.

If the current node is not the same is the source node of the flow, this means that there is already a path stored in the path_matrix. The following lines scan the flow-index-th row for the current node to find its neighbour, which is the next hop.
For (i=1; i<= number_of_nodes; i++)
{
  if (path_matrix[flow->index][i]-1==node_conidered)
  {
    sprintf(next_node,"node%d", path_matrix[flow->index][i-1]-1);
  }
}

Example:
Given the following:
Number of flows = 2
Number of nodes = 5
Current active node = 4
Path_matrix = [2 3 1 0 0 ; 2 4 3 1 0]
When flow 2 is considered, the second row of the matrix is scanned in search for
the value of node_considerd – 1, which is in this case equal to 3. So it is clear now
that the next hop is equal to 4-1=3. Remember that the path calculated by Samcra
goes from source to destination! Note also that a ‘0’ in the path_matrix does not
refer to ‘node0’, but indicates a free space in the matrix. Integer ‘1’ is equal to
‘node0’!

The next steps are identical to that in case of Samcra hh.

Part V
3.2.4.1 Difference of path between Samcra hh and sb

As the path is pre-calculated in case of Samcra source-based, it is acceptable to
believe that the path could be different than the case when Samcra hop-by-hop is
used. It is interesting to monitor these changes of the path.
In our definition it does not matter how big the difference is between two paths in
terms of hops. So if the number of different hops along a path of a flow is equal to
or greater than 1, we regard that as ‘one’ difference.
For this we define the ‘dif’ vector. There is a maximum number of flows of 30 and
we need one more additional element in the vector so ‘dif’ is defined as a vector
with 31 elements.
Here the idea is explained in case there are 5 flows:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>....</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>....</td>
<td>0</td>
</tr>
</tbody>
</table>

The values of the first row are used in the figure to indicate the position of the
elements.
The initial value of all the elements is zero.
Each time after the calculation of the path and the next hop by Samcra, it is checked whether the next hop value is the same as the next hop in the path matrix. If not the flow-index-th value of the ‘dif’ vector is checked for its value. If it contains a ‘1’, it means that a difference in the path of the current flow has already been registered and need not to be registered again. If the value is equal to ‘0’, it means that this is the first time that a difference has occurred and the value of the flow-index-th element is set to ‘1’.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>….</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>….</td>
<td>0</td>
</tr>
</tbody>
</table>

This means that the flow 4, has at least one hop different than the case of hop-by-hop path calculation.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>….</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>….</td>
<td>1</td>
</tr>
</tbody>
</table>

Every time that the current node is equal to the source node of the flow, the flow-index-th value of the ‘dif’ vector is checked. If this value is equal to ‘1’, the value of dif[31] will be add by ‘1’ and the flow-index-th value in the ‘dif’ vector will be reset.

At the end of a simulation, the value of dif[31] is printed in a text file named ‘difference’.

3.2.5 Additional codes to record the paths and costs

In order to study the behaviour of the algorithms and make a better comparison with other ones, some code has been added, which enable the user to see the path, the cost of the path and the number of hops for each flow during a simulation. This information is stored in a file called ‘Total_cost_of_routingalgorithm’. The code is implemented at the end of the code of each of the four routing algorithms. The result printed in the output file has the following structure:

The path
Flow index number of hops the total cost of the path.

When two simulations have been run, the results of both of them will be added to this file and are separated by a dashed line like ‘--------’.
Part III

4. Link state update

As mentioned in the explanation of the QOSPF component in paragraph 2.2.5, there are four types of link state update policies in QRS1.1, which are all implemented in the file qospf.c.

The policies are defined in the function \textit{rm\_gospf}(g, flow), which is relatively at the end of qospf.c.

The implementation is straight and with the information given in previous chapters about different components of QRS1.1, it should be possible to follow the code. As the thesis was merely about using this policies and not modifying them, a deeper understanding of the implementation of this part of the program is left to the reader.
References


