Modeling a shared medium access node with QoS distinction

Matthias Gries, Jonas Greutert
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology Zürich
CH-8092 Zürich, Switzerland
email: {gries,greutert}@tik.ee.ethz.ch

TIK-Report No. 86
March 30, 2000

Abstract

This report describes a high-level design space exploration for the implementation of an IP over ATM shared-medium access node with Quality of Service (QoS) distinction and routing functionality. The different blocks of the architecture are modeled with different optimization criteria in mind. The models provide throughput, delay, costs, and memory space worst-case bounds depending on a variety of parameters and implementation alternatives. We then reveal different optimal designs for the main memory system of the access node and point out the hot spots of the architecture and implementation. As a result, a node supporting a 155 Mbit line rate, a state of the art fair queuing scheduler with up to 2^{13} concurrent connections, a buffer for at least 2^{15} packets, and a backbone router with up to 40000 routing entries can be implemented using a single moderately clocked general purpose CPU core with two memory buses and four to eight memory chips. The resources show a worst-case utilization of 75 % in terms of latency, so that there is room for further building blocks and higher line rates.

Contents

1 System overview

The modeled network access node is part of a network which uses a line topology (see Fig. 1). ATM connections are established for bundles of IP flows with the same source and destination node on the line. Thus, the ATM line forms a transparent backbone for IP flows. Incoming UBR traffic is always taken out from the line at the input of a node and fed in the ATM traffic at the output of a node, provided that the node is not the destination of the UBR traffic. All other kinds of traffic can pass through an ATM node on a direct data path with highest priority to minimize the latency introduced by the node for line traffic. The line access arbitration is implemented with a frame slot reservation scheme. In order to keep the management reasonably simple as well as access latencies short, the number of slots and the frame length are fixed. That is, a 100 % usage of the line with non-UBR traffic can only be achieved using particular rates. For an arbitrary flow rate, line bandwidth must be overbooked reducing the usage of the line. However, a 100 % usage of the line can still be reached by stuffing unused bandwidth with UBR traffic at the line access level.

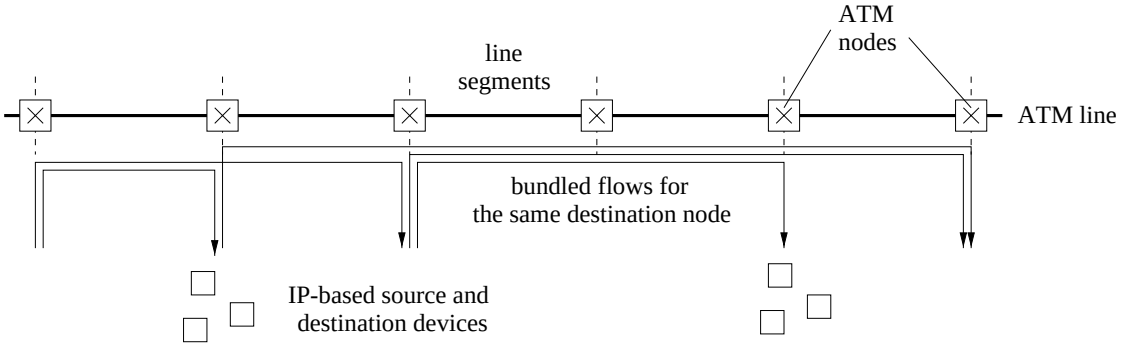

Figure 1: System overview.

The inner architecture of a line node is displayed in Fig. 2. IP flows are classified, routed, and scheduled on a packet basis. The ATM cell segmentation is performed afterwards. For each destination node there is a scheduler which bundles IP flows according to either fixed priority or individual QoS classes. The modeled architecture only considers components for a static network scenario, i.e., ATM connections are established and frame slots have been assigned to connections. The models in particular underpin the influence of the main memory. Therefore, models for recent RAM technologies (SDRAM, RDRAM, SRAM) have also been taken into account.

Modeling methodology

The characteristics of the building blocks of the access node, such as memory space and memory throughput requirements, calculations, and introduced delay have been determined by the analysis of suitable algorithms. The following assumptions have been made: only arithmetic operations have been counted, no branch or loop instructions have been counted. Instructions are found in on-chip cache memory and thus do not generate additional delay due to fetching. Moreover, allocation and deallocation of memory segments only need one further operation. Since only memory segments of fixed size are required, the free memory list can be implemented by a linked list together with distinct memory areas for the list and the memory segments. Moreover, memory fragmentation does not affect system performance. In this configuration, only a couple of additional registers are required to keep track of the FILO-organized

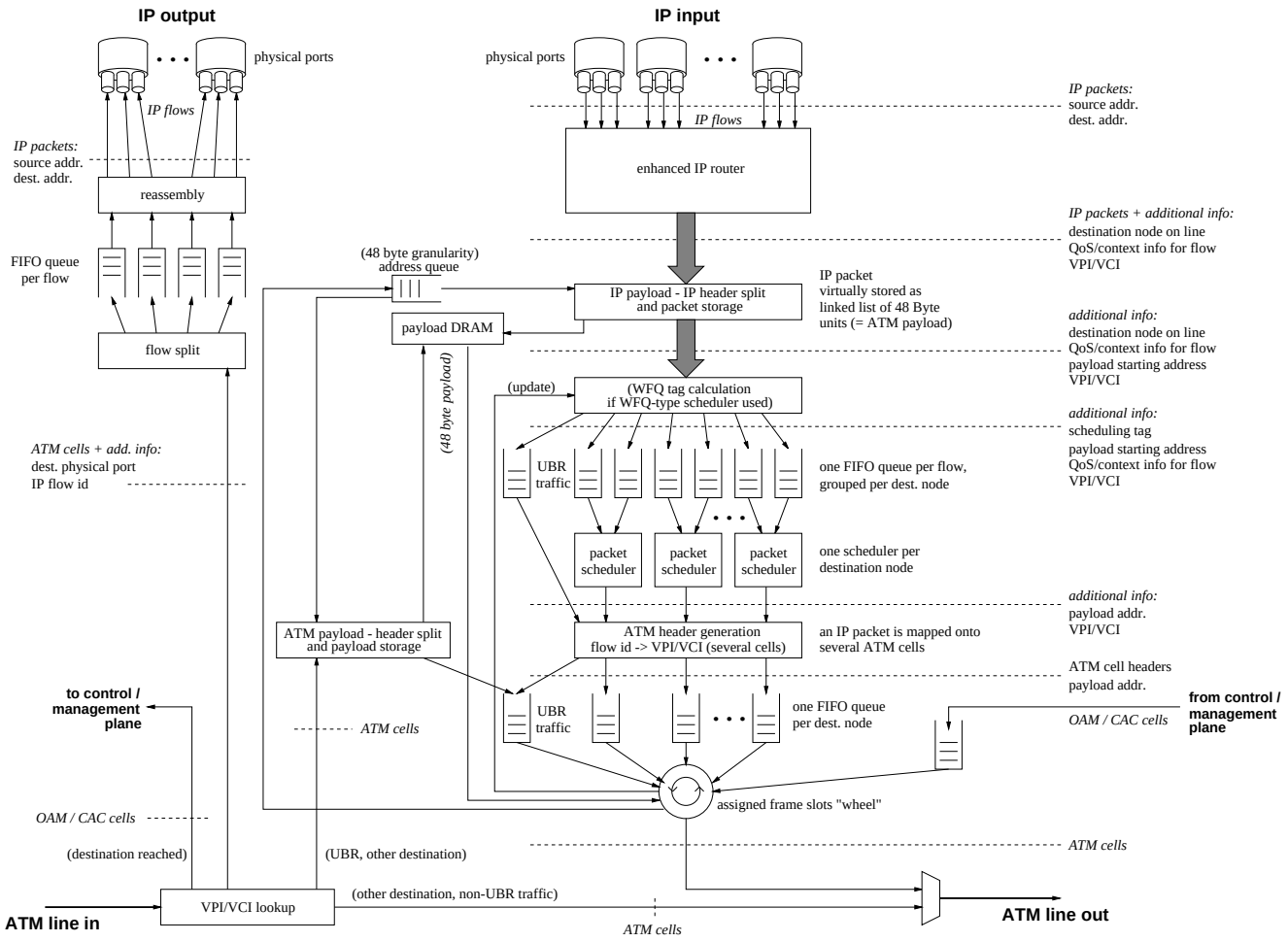

Figure 2: ATM node architecture: IP layer + ATM Plane / ATM Layer.

free list. The delay analysis considers three kinds of reasons: delay introduced by memory accesses, by calculations, and by algorithm characteristics such as priority inversion if packet scheduling is performed. In the worst-case, data dependencies prevent the concurrent execution of calculations and memory accesses. Therefore, the delay bounds for the three cases have been added. In the given examples, two different memory configurations have been investigated: an SDRAM running at 100 MHz with an average access overhead of four cycles and an SRAM running at 166 MHz with two cycles access overhead. Both memory subsystems use a 32 bit data bus. Finally, the throughput analysis shows whether the chosen memory technology is suitable for the corresponding task. The reciprocal of the delay bound divided by the size of an ATM cell (IP packet respectively) should be less than the rate that has to be supported. Otherwise, the system can be overloaded.

The results of this report can thus be used to estimate the impact of a variety of parameters such as ATM rate, number of connections, packet sizes, etc. on the requirements of the memory subsystem and the needed computing power without the help of any special hardware functional units. The analysis can be performed independently for the building blocks of the node, e.g. for a pipelined implementation of the system by a chain of processor modules. The case study in this report however shares a single

processor module in order to implement all building blocks of the access node.

This report is organized as follows: in the next section, different main memory models are developed taking simulations of real-world applications into account. Section 3 summarizes the common modeling parameters used in the succeeding sections. Sections 4 to 11 describe the models for the different building blocks of the access node. Each of these sections provides examples for typical system settings. Section 12 finally shows optimal configurations of the main memory system in terms of costs and latency for an access node considering the models of the preceding sections. Section 13 concludes the report.

2 Memory Model

The models for main memory accesses derived in this section are used in subsequent sections for the determination of realistic throughput and delay bounds for the building blocks of the access node.

Nomenclature

g	$[bit]$	granularity of data, i.e., smallest amount of data to be accessed and transferred by a burst
f	$[MHz]$	clock frequency of the memory bus
cap	$[bit]$	capacity of a single memory chip
db	$[bit]$	data bus width of the memory chip
b		number of memory banks within a chip
gd	$[bit]$	size of the data structure to be stored
c	[$]	costs per chip
bb	$[bit]$	width of the memory bus
cmc	[$]	costs of the memory controller
m		number of memory modules
bc		number of independent memory buses
ov	$[cycles]$	overhead for processing a read or write request dependent on memory type, e.g, an SDRAM may need up to six extra clock cycles for preparing a burst transfer.
ps	$[bit]$	Rambus RAM specific parameter: size of a data packet

Simplifications

- no distinction between read and write accesses and no consideration of the internal state of the RAM. That is, all transfers need the same latency.
- refresh cycles are ignored.
- the smallest amount of data to be accessed fits in a memory row.
- an ideal interleave of memory banks and modules is assumed. In reality, data dependencies usually prevent this behavior (see subsec. 2.5).

2.1 Model for a single chip with bank interleave

A single memory transfer without interleaving is shown in Fig. 3 a). The SDRAM needs $\lceil \frac{g}{db} \rceil$ clock cycles for the transfer of the data. Moreover, ov cycles must be spent for preparing the transfer. Thus, a single burst transfer needs $ov + \lceil \frac{g}{db} \rceil$ cycles for completion. However, the ov cycles can be hidden if more than one memory bank is available within the chip. Fig. 3 b) shows two transfers using two memory banks hiding some of the overhead cycles. Finally, if enough memory banks are available, the overhead cycles can be hidden completely as shown in Fig. 3 c).

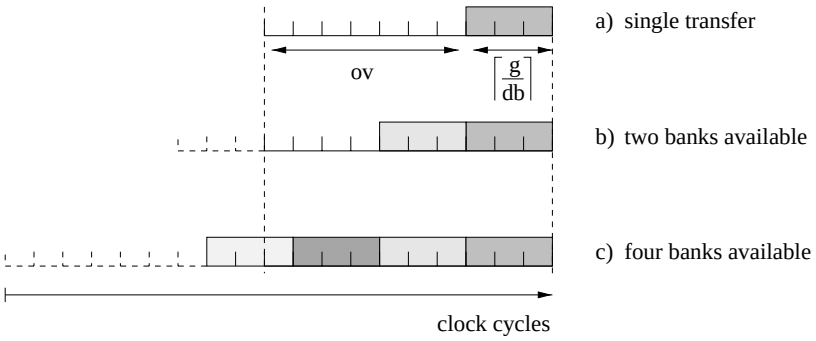

Figure 3: Memory bank interleave using an SDRAM.

- throughput th:

Parameter	SDRAM	RDRAM [a)]	SRAM
X	$\lceil \frac{g}{db} \rceil$	$\lceil \frac{g}{ps} \rceil \cdot \frac{ps}{db \cdot 2}$	$\lceil \frac{g}{db} \rceil$

a) *The RDRAM uses both edges of the clock.*

$$th = \frac{\#\text{bits transferred}}{\text{cycles needed}} = \frac{b \cdot g}{\max\left(b \cdot X, ov + X\right) \cdot \frac{1}{f}} \tag{1}$$

- costs:

$$\left\lceil \frac{gd}{cap} \right\rceil \cdot c + cmc$$

2.2 Several chips forming a wider memory bus

In this model, several memory chips form a wider memory bus. All chips are controlled by the same signals. For instance, using a single SIMM or DIMM memory module is an example for this configuration. RDRAMs are usually not used in this configuration.

- throughput th for SDRAMs / SRAMs:

$$th = \frac{b \cdot g}{\max\left(b \cdot \lceil \frac{g}{bb} \rceil, ov + \lceil \frac{g}{bb} \rceil\right) \cdot \frac{1}{f}} \tag{2}$$

- costs:

$$\left\lceil \frac{gd}{\frac{bb}{db} \cdot cap} \right\rceil \cdot \frac{bb}{db} \cdot c + cmc$$

2.3 Wide memory bus with several modules

In this model, several memory chips form a wider memory bus. All chips are controlled by the same signals. Several memory modules can be controlled concurrently sharing the same memory bus. Using

several SIMMs or DIMMs on a single memory bus is an example for this configuration when using SDRAMs or SRAMs. For RDRAMs ($bb = db$), several memory chips are controlled concurrently using the same bus. The number of needed memory modules can be determined from the size of the data structure gd or be given by the parameter m to enforce a higher degree of parallelism. Basically, increasing the number of memory modules provides more individual banks in this model.

- throughput th:

Parameter	SDRAM	RDRAM [a)]	SRAM
X	$\lceil \frac{g}{bb} \rceil$	$\lceil \frac{g}{ps} \rceil \cdot \frac{ps}{db \cdot 2}$	$\lceil \frac{g}{bb} \rceil$

a) *The RDRAM uses both edges of the clock and* $bb = db$.

$$th = \frac{\max\left(m, \left\lceil \frac{gd}{\frac{bb}{db} \cdot cap} \right\rceil\right) \cdot b \cdot g}{\max\left(\left\lceil \frac{gd}{\frac{bb}{db} \cdot cap} \right\rceil \cdot b \cdot X, m \cdot b \cdot X, ov + X\right) \cdot \frac{1}{f}} \tag{3}$$

- costs:

$$\max\left(\left\lceil \frac{gd}{\frac{bb}{db} \cdot cap} \right\rceil, m\right) \cdot \frac{bb}{db} \cdot c + cmc \tag{4}$$

2.4 Several separate memory buses

Using several memory buses concurrently simply means multiplying the cost and throughput measures in subsection 2.3 by the number of buses bc. Note that a separate memory controller is needed for each bus.

2.5 Memory model parameters derived from simulations

In this subsection, average values for the memory access overhead (ov) and the influence of bank interleaving are determined. Simulations described in [13] have shown that the speedup of the memory system is clearly less than a factor of two by applying bank interleaving due to data dependencies and the amount of branches in real-world applications. The average overhead ov for the application mix and CPU configuration used in [13] has been determined as follows, see Tab. 1.

These results suggest to simplify the formula given in subsections 2.1 and 2.2. The effect of using several modules as described in subsection 2.3 has not been investigated in [13]. Equation 1 reduces to

$$th = \frac{g}{(ov + X) \cdot \frac{1}{f}}$$

by removing the influence of the number of banks b in favor of using the corresponding value of the memory access overhead ov given in Tab. 1. Similarly, equation 2 reduces to

$$th = \frac{g}{(ov + \lceil \frac{g}{bb} \rceil) \cdot \frac{1}{f}}$$

SDRAM, four internal banks, 100 MHz bus	average overhead *ov* [ns]/[cycles]
closed-page policy	41.5 / 4.15
open-page, no bank interleaving	36.8 / 3.68
open-page, bank interleaving	30.1 / 3.01

RDRAM, 32 internal banks, 400 MHz bus	average overhead *ov* [ns]/[cycles]
closed-page policy	51.1 / 20.44
open-page, no bank interleaving	46.7 / 18.68
open-page, bank interleaving	31.3 / 12.52

Table 1: Memory access overhead dependent on DRAM and controller policy used.

3 Common parameters for the building blocks of the access node

The following parameters will be used in the succeeding sections to model the different building blocks of the access node as shown in Fig. 2.

N		number of nodes along the ATM line
cs	$[bit]$	ATM cell size
hs	$[bit]$	ATM cell header size
r	$[bit/s]$	rate of the ATM line
clk	$[MHz]$	CPU clock frequency
$ipps_{max}$	$[bit]$	maximal IP packet size
$ipps_{min}$	$[bit]$	minimal IP packet size
vis	$[bit]$	VPI/VCI field size
con	$[bit]$	space needed for storing the context information of a connection, i.e., the max. packet length and the reserved (minimal) rate.
nc		max. number of connections per ATM node

4 The "wheel" framing mechanism

It is assumed that connections with the same destination node share a frame slot.

Specific parameters

fs	$[slots]$	number of cell slots in a frame
$addr_{fs}$	$[bit]$	number of address bits needed for random accesses in the memory used for the frame implementation (can be calculated from the size of the data structure gd and the memory chip organization)
$addr_{hd}$	$[bit]$	address bits for header queue accesses; the memory for frame and header data structures may be shared.
lat_{wh}	$[s]$	average memory access latency for wheel setting

4.1 Requirements for the frame storage

Each frame entry consists of a source node id and a destination node id. By the source id the node identifies reserved frames. A comparison of $\lceil \log_2 N \rceil$ bit values is needed for this task in each cell clock cycle. The destination id then determines the corresponding FIFO queue from which a cell can be put on the ATM line.

- space required: $fs \cdot 2 \cdot \lceil \log_2 N \rceil$
  (source + destination node id) bits of memory space for frame storage.
- throughput needed : $\frac{r}{cs} \cdot 2 \cdot \lceil \log_2 N \rceil$
  for reading out frame positions in each cell cycle.
- calculations for a cell: two operations
  A comparison of $\lceil \log_2 N \rceil$ bit values and a frame address increment.
- delay for a cell: $2 \cdot (lat_{wh} + clk^{-1})$ with $g = max(\lceil \log_2 N \rceil, addr_{fs}, addr_{hd})$.

4.2 Requirements for determining the destination queue

4.2.1 p-trie implementation

A complete multiway trie [18] with p pointers in each node needs $\lceil \log_p N \rceil$ address offset calculations for determining the destination queue of a frame slot.

- space worst-case: $\frac{1}{p-1}(p^{\lceil \log_p N \rceil} - 1) \cdot p \cdot addr_{fs} + N \cdot addr_{hd}$
  The first term considers the space needed for the pointers at the next nodes in the trie and the second one stands for the pointers to the FIFO queues.
- throughput needed for traversing the trie: $\frac{r}{cs} \cdot \left(\lceil \log_p N \rceil \cdot addr_{fs} + addr_{hd} \right)$
- calculations: $\lceil \log_p N \rceil$ address offset calculations per cell clock cycle
- delay for a cell: $(\lceil \log_p N \rceil + 1) \cdot lat_{wh} + \lceil \log_p N \rceil \cdot clk^{-1}$

4.2.2 Table lookup implementation

- space required: $N \cdot addr_{hd}$
- throughput needed for reading out the table: $\frac{r}{cs} \cdot addr_{hd}$
- calculations: an offset address calculation per cell clock cycle
- delay for a cell: $lat_{wh} + clk^{-1}$

4.3 Example

See Tab. 2.

5 Destination queues and IP-ATM header conversion

The ATM cell header data and the corresponding payload addresses are stored after they have passed the schedulers.

Fixed model parameters

addr	20 bit	addresses for header and frame data structures
cs	424 bit	cell size
clk	200 MHz	CPU clock

Model parameters				*4-trie implementation*				*Lookup implementation*			
N	*fs*	*r*	lat_{wh}	*sp*	*th*	*ops*	*del*	*sp*	*th*	*ops*	*del*
		[Mbit/s]	[ns]	[Kbit]	[Mbit/s]	[10^6]	[ns]	[Kbit]	[Mbit/s]	[10^6]	[ns]
256	1024	155	50	27.6	40.4	2.2	380	21.0	12.6	1.0	165
256	1024	155	20	27.6	40.4	2.2	170	21.0	12.6	1.0	75
256	1024	622	50	27.6	162.3	8.8	380	21.0	50.4	4.2	165
256	1024	622	20	27.6	162.3	8.8	170	21.0	50.4	4.2	75
256	2048	155	50	43.6	40.4	2.2	380	37.0	12.6	1.0	165
256	2048	155	20	43.6	40.4	2.2	170	37.0	12.6	1.0	75
256	2048	622	50	43.6	162.3	8.8	380	37.0	50.4	4.2	165
256	2048	622	20	43.6	162.3	8.8	170	37.0	50.4	4.2	75
512	1024	155	50	54.6	48.1	2.6	435	28.0	13.2	1.0	165
512	1024	155	20	54.6	48.1	2.6	195	28.0	13.2	1.0	75
512	1024	622	50	54.6	193.1	10.3	435	28.0	53.2	4.2	165
512	1024	622	20	54.6	193.1	10.3	195	28.0	53.2	4.2	75
512	2048	155	50	72.6	48.1	2.6	435	46.0	13.2	1.0	165
512	2048	155	20	72.6	48.1	2.6	195	46.0	13.2	1.0	75
512	2048	622	50	72.6	193.1	10.3	435	46.0	53.2	4.2	165
512	2048	622	20	72.6	193.1	10.3	195	46.0	53.2	4.2	75

Table 2: Wheel framing example.

Specific parameters

$addr_{pl}$	$[bit]$	number of address bits needed for random accesses in the memory used for the payload storage
$addr_{hd}$	$[bit]$	address bits for header queue accesses
lat_{pl}	$[s]$	average memory access latency for payload RAM
lat_{hd}	$[s]$	average memory access latency for ATM header queue RAM
lat_{ip}	$[s]$	average memory access latency for packet header queue RAM

5.1 Requirements of the destination queues

Since the queues are positioned after the WFQ schedulers and overbooking is thus not possible, at most one packet must be buffered for each destination, i.e., $\lceil \frac{ipps_{max}}{cs-hs} \rceil$ ATM cells must be stored per destination in the worst-case. The framing scheme, introducing a maximal delay of two ATM cell periods, requires that at least two ATM cells must be buffered per destination node. This condition is always satisfied if the minimal packet size is larger than 48 byte.

- space required: $(N-1) \cdot \lceil \frac{ipps_{max}}{cs-hs} \rceil \cdot (hs + addr_{pl})$
  However, the space required for UBR traffic may be arbitrarily chosen. The IP packet payload remains unchanged in the shared memory.

- throughput required: $\frac{r}{cs}(addr_{pl} + hs)$
  is needed for the read out of the headers by the wheel. The throughput $\frac{cs-hs}{cs} \cdot r$ needed for reading

out the payload (using separate chips for header and payload) will be considered in section 7.

- calculations: memory entry for header is freed.
- delay for a cell: $lat_{hd} + clk^{-1}$ for a cell header read from header RAM.
  The granularity g is $addr_{pl} + hs$. The payload RAM will be considered in section 7.

5.2 Requirements of the header conversion

The packet schedulers determine which IP packet has to be converted next. The corresponding information (VPI/VCI and payload addresses) is read from the packet queues before the schedulers and copied several times (dependent on the packet length) into the corresponding destination queue.

- throughput demand of the packet queues (considered in section 6): $\frac{\frac{r}{cs}}{\lceil \frac{ipps_{min}}{cs-hs} \rceil}(vis + addr_{pl})$
- throughput demand of the destination queues: $\frac{r}{cs}(hs + addr_{pl})$
- memory space needed: during IP-ATM header conversion only a single ATM header must be stored which is used as template for the ATM header generation. Usually, registers are used for this purpose. The space needed in the packet and destination queues is already considered in the corresponding subsections.
- calculations: memory entry for the header must be allocated. No additional calculations are needed because the assignment of schedulers to destination queues is fixed.
- delay: $lat_{hd} + clk^{-1}$ per cell for writing the header into a header queue, lat_{ip} per packet for reading out a packet queue entry (considered in section 6). For the packet queues, g is $max(ts + addr_{pl} + vis + con, addr_{ip})$ in the WFQ case and $addr_{pl} + vis$ for the fixed priority scheduler respectively.

5.3 Example

See Tab. 3.

6 Packet scheduler

Due to quality of service (QoS) distinction requirements in access and core networks, QoS must be distinguished for groups of flows (DiffServ approach [16]) classified by traffic characteristics, applications, organizations, or protocols and even for individual flows (IntServ approach [16]). Scheduling algorithms for output link sharing must therefore be able to assign link bandwidth to classes of flows independently on the behavior of other flows. The idea to adapt the behavior of an ideal fluid server to the time-multiplex in packet networks has been the basis for a variety of packet scheduling algorithms and has been used first by Weighted Fair Queuing (WFQ) [9, 23]. An ideal fluid server is able to serve several flows concurrently according to weights. Excess bandwidth not reserved by any flow is distributed in a fair manner on backlogged flows again according to the assigned weights. A WFQ server then schedules packets according to the order in which they would finish service in a fluid system. For this purpose a WFQ system labels each incoming packet by the time or amount of normalized service at which the packet would leave the fluid server. The arriving packets are sorted according to these scheduling tags in increasing order and the scheduler chooses the packet with the minimum scheduling tag for transmission over the output link. It has been shown that by applying WFQ the packet system can only be a packet

Fixed model parameters		
cs	424 bit	cell size
hs	40 bit	cell header size
$ipps_{max}$	1536 byte	maximal packet size
$ipps_{min}$	40 byte	minimal packet size
vis	28 bit	VPI/VCI field size
$addr_{pl}$	22 bit	payload addresses
clk	200 MHz	CPU clock

Model parameters			*dest. queues and header conv.*			
N	r	lat_{hd}	sp_{hd}	th_{hd}	ops	del
	[Mbit/s]	[ns]	[Kbit]	[Mbit/s]	[10^6]	[ns]
256	155	60	505.9	43.2	0.7	140
256	155	25	505.9	43.2	0.7	50
256	622	60	505.9	173.5	2.9	140
256	622	25	505.9	173.5	2.9	50
512	155	60	1046.5	43.2	0.7	140
512	155	25	1046.5	43.2	0.7	50
512	622	60	1046.5	173.5	2.9	140
512	622	25	1046.5	173.5	2.9	50

Table 3: Destination queues and IP-ATM header conversion example.

length behind the service of the ideal fluid system in the worst-case. However, there is a tremendous overhead for the calculation of the scheduling tags in WFQ because a WFQ scheduler must internally emulate a fluid server in order to calculate finish times or service levels. Moreover, events can appear frequently in the fluid system since virtually all backlogged flows may finish service at the same time. In order to find a suitable trade-off between the complexity of the scheduling algorithm, the fairness for the distribution of excess bandwidth, and the provision of sharp delay bounds, different approaches have been applied in order to implement WFQ.

Self-clocked fair schedulers So-called *self-clocked* methods no longer emulate a fluid system but estimate scheduling tags by the tags of packets which are currently queued in the packet system. Self-clocked fair queueing (SCFQ) [11] uses the finish time of the packet currently in service for the estimation of the state of the fluid server. Contrary to that, start-time fair queuing (SFQ) as described in [12] uses the start time of the packet currently in service for the estimation and serves packets in increasing order of their start times. Minimum starting-tag fair queueing (MSFQ) [6] serves packets in increasing order of their finish times. The state of the fluid server is estimated by the minimum of the start times of backlogged flows. A second priority queue is therefore required. The same approach has been independently published in [7] under the name time-shift scheduling. Self-clocked algorithms decrease the implementation complexity of WFQ but usually provide worse delay bounds.

Approximation by potential functions More methodical approaches of fair queuing designs are based on the theory of rate-proportional servers (RPS) [31]. A scheduler of the type RPS keeps track of the state of the fluid system by a system potential function (often also called virtual time or service). Scheduling tags can then be seen as the level of the system potential function normalized to the weight of the corresponding flow at which the packet would depart the fluid system. Algorithms which do not

emulate the fluid server precisely use a so-called base potential function in order to recalibrate the system potential at certain points of time. The system potential then increases linearly between recalibrations (assuming the system is not idle). Different base potential functions as well as different recalibration time intervals can be chosen in order to find a suitable trade-off between fairness and complexity of the scheduler. RPS based schedulers achieve the same worst-case delay bounds as WFQ. Starting potential-based fair queueing (SPFQ) and frame-based fair queueing (FFQ) have been presented in [30]. SPFQ recalibrates the system potential at every packet departure. The base potential is updated at every packet arrival and is set to the minimum start potential of all backlogged flows, that is, the potential at which the corresponding packet would begin getting service in the fluid system. Thus, SPFQ is very similar to MSFQ. MSFQ however does not use a system potential and recalibrates at packet arrivals. FFQ uses a simpler base potential than SPFQ and larger intervals between recalibrations at the expense of fairness. Minimum delay self-clocked fair queueing (MD-SCFQ) [4] uses the same recalibration intervals as SPFQ together with a simplified base potential which does not need to maintain a second priority queue in order to manage start potentials. However, MD-SCFQ can achieve better fairness than SPFQ for certain flow settings.

Eligible packet selection Although the amount of service by which a WFQ system can be behind a fluid system is bounded, the WFQ system schedule can be quite ahead of the fluid system. This behavior shows undesired properties if feedback congestion control is used e.g. for the regulation of best-effort traffic [2]. In order to retain fairness between the flows sharing a link not only on the average but also on a fine time granularity, there are scheduling algorithms which use two distinct priority queues for sorting. Worst-case fair weighted fair queueing (WF^2Q) [2] and its more efficient implementation WF^2Q+ in [1] sort arriving packets according to their start time in the fluid system. Only packets which are eligible for transmission, that is, for which service would have been started in the fluid system, are then transferred to the second priority queue which is sorted according to the finish times. WF^2Q+ does not need to emulate a fluid server but uses an approximation similar to SPFQ and MSFQ. However, WF^2Q+ considers only eligible packets for transmission. Leap Forward Virtual Clock [33] transfers packets from backlogged but oversubscribed flows to a second priority queue. Packets residing in this queue are not eligible for transmission yet. Care is taken that packets are written back to the first priority queue before any delay bound may be missed. Both algorithms have in common that the full contents of one priority queue must be copied to the other priority queue between two scheduling decisions in the worst-case.

Round-Robin variants There are scheduling algorithms with very low complexity which enhance the concept of a Round-Robin scheduler with virtual service ideas. However, the algorithms presented in [5] and [27] cannot provide sharp delay or fairness bounds as schedulers which use a fluid server as reference model.

Hierarchical grouping If a single level of flows or flow classes is not detailed enough in order to distinguish QoS, one may think about using several levels of schedulers within a hierarchy [1, 10, 32]. However, the delay and fairness properties of the schedulers are accumulated through the levels of the hierarchy. Moreover, in the latter case, where a scheduler based on service curves is used [26], one should be aware of the complexity overhead involved by applying service curves which model more complex behavior than leaky bucket constraint sources.

Modeled schedulers The scheduler stage consists of the scheduling tag calculation, enqueue and dequeue operations on a priority queue as well as context information updates of the virtual service

measure. Special hardware such as [3, 24, 20] which accelerates the operations of priority queues under specific constraints has not been considered since we are interested in an implementation on general purpose, programmable hardware. We therefore focus on the influence of a data structure for priority queues proposed in [29] on the performance of the schedulers. The original weighted fair queuing (WFQ) algorithm [23], a self-clocked variant MD-SCFQ [4], and a fixed priority scheduler are considered. We restrict our study on algorithms which only need a single priority queue. It is assumed that the bandwidth of the output link is not overbooked.

Specific parameters

r_{min}	$[bit/s]$	minimally reservable rate in a WFQ system
r_{max}	$[bit/s]$	maximally reservable rate in the WFQ approximation system
pcs		number of priority classes (fixed priority scheduler only)
npc_{p_i,d_i}		number of elements currently stored in the corresponding priority class queue with priority p_i and destination node d_i (fixed priority scheduler only)
$addr_{ip}$	$[bit]$	number of address bits needed for random accesses in the memory used for the IP header storage
$addr_{ss}$	$[bit]$	number of address bits needed for random accesses in the memory used for the scheduling tag storage
$addr_{pl}$	$[bit]$	number of address bits needed for random accesses in the memory used for the payload storage
lat_{ip}	$[s]$	average memory access latency for packet header RAM
lat_{ss}	$[s]$	average memory access latency for scheduling tag RAM
ts	$[bit]$	scheduling tag precision
pn_{d_i}		number of elements currently stored in the corresponding QoS queues for destination node d_i. (WFQ/WFQ approx. scheduler only)
cp	$[bit]$	backlog counter precision (WFQ/WFQ approx. scheduler only)
$nsic$		number of service interval classes (WFQ approximation only)
x		number of interval subdivisions (WFQ approximation only)

6.1 Fixed Priority scheduler

6.1.1 Enqueue

Since priority classes are statically determined for each IP flow, packet scheduling tag calculations are not required for this scheduler. However, the packet must still be appended to the corresponding FIFO queue which collects packets of all flows of a particular priority class with the same destination node. Two possible implementations for this lookup are considered here. Simplification: additional operations and data structures required if a queue becomes backlogged or empty are not considered.

- lookup implemented as 2D-array: the lookup table is organized as a 2D-array of pointers using the destination node and the priority class as indices. The pointers direct to the end of the corresponding FIFO queue.
  - space required: $(N-1) \cdot pcs \cdot addr_{ip}$
  - throughput: $\frac{r}{ipps_{min}} \cdot addr_{ip}$
  - calculations: one 2D address offset calculation for every packet
  - delay per packet: $clk^{-1} + lat_{ip}$

- lookup implemented as 2D-vector: the lookup table is organized as a vector of pointers. The destination node is used as index. each pointer may point to another vector of pointers that is indexed by the priority class. The pointers in these vectors direct to the end of the corresponding FIFO queue.
  - worst case space required: $(N-1) \cdot addr_{ip} \cdot (1 + pcs)$
  - throughput: $\frac{r}{ipps_{min}} \cdot 2 \cdot addr_{ip}$
  - calculations: two address offset calculations for the vector accesses for every packet are needed.
  - delay per packet: $2 \cdot (clk^{-1} + lat_{ip})$ with $g = addr_{ip}$
- FIFO queues for the different priority classes (linked lists):
  - space: $\sum_{p_i,d_i} npc_{p_i,d_i} \cdot (addr_{ip} + addr_{pl} + vis)$
  - throughput: $\frac{r}{ipps_{min}} \cdot (2 \cdot addr_{ip} + addr_{pl} + vis)$
  - calculations: allocate memory for new node and adjust two pointers for every packet.
  - delay per packet: $clk^{-1} + lat_{ip}$ with $g = addr_{ip} + addr_{pl} + vis$

6.1.2 Dequeue

Simplification: additional operations and data structures required if a queue becomes backlogged or empty are not considered. That is, all backlogged queues remain backlogged after a dequeue operation.

- space: already considered at the enqueue operation
- throughput: $\frac{r}{ipps_{min}} \cdot (2 \cdot addr_{ip} + addr_{pl} + vis)$
- calculations: deallocate memory, adjust a pointer for every packet
- delay per packet:
  - introduced by the scheduling algorithm: $d_m = \frac{ipps_{max} \cdot (1 + \sum_{p=1}^{m} \sum_{j \in S_p} 1)}{r \cdot (1 - \sum_{p=1}^{m-1} \mu_p)}$
    worst-case delay for a packet in the class of priority m using the bounds given in [37], assuming a constant inter-arrival time for packets of variable length (i.e. $X_{ave} \equiv X_{min}$ in [37]), with S_p the set of flows at priority level p, r_i the minimal rate of flow i, with $\mu_p = \frac{ipps_{max}}{ipps_{min} \cdot r} \cdot \sum_{j \in S_p} r_j$ and $\sum_{p=1}^{pcs} \mu_p \leq 1$ (schedulable setting).
  - introduced by calculations: clk^{-1}
  - introduced by memory accesses for every packet: lat_{ip} with $g = addr_{ip} + addr_{pl} + vis$

6.1.3 Example

See Tab. 4. The column $del.fix$ summarizes the delay introduced by calculations and memory accesses. The column $del.variation$ stands for the worst-case delay bounds provided by fixed priority scheduling for the highest priority IP flow compared with the lowest priority flow. The column sp_{vec} only considers the amount of memory needed for the vector lookup. The memory space required for IP headers depends on the amount of UBR traffic and the characteristics of the IP flows such as burst lengths. For instance, supporting up to 2^{13} IP flows with an average of two maximum sized packets queued in the system requires 1.2 Mbit of IP header storage.

Fixed model parameters

$ipps_{max}$	1536 byte	maximal packet size
$ipps_{min}$	40 byte	minimal packet size
vis	28 bit	VPI/VCI field size
$addr_{pl}$	22 bit	payload addresses
$addr_{ip}$	22 bit	packet header addresses
clk	200 MHz	CPU clock
nc	2^{13}	max. number of connections per ATM node

Model parameters						*vector lookup, FPS*				
N	r	$lat_{ip,addr}$	+	lat_{ip}	pcs	sp_{vec}	th	ops	$del.fix$	$del.variation$
	[Mbit/s]	[ns]				[Kbit]	[Mbit/s]	[10^6]	[ns]	[ms]
256	155	50	+	70	10	60.3	107.2	1.94	260	0.16 - 6500
256	155	50	+	70	64	356.1	107.2	1.94	260	0.16 - 6500
256	155	20	+	30	10	60.3	107.2	1.94	120	0.16 - 6500
256	155	20	+	30	64	356.1	107.2	1.94	120	0.16 - 6500
256	622	50	+	70	10	60.3	107.2	1.94	260	0.04 - 1600
256	622	50	+	70	64	356.1	107.2	1.94	260	0.04 - 1600
256	622	20	+	30	10	60.3	107.2	1.94	120	0.04 - 1600
256	622	20	+	30	64	356.1	107.2	1.94	120	0.04 - 1600
512	155	50	+	70	10	120.8	430.1	7.78	260	0.16 - 6500
512	155	50	+	70	64	713.6	430.1	7.78	260	0.16 - 6500
512	155	20	+	30	10	120.8	430.1	7.78	120	0.16 - 6500
512	155	20	+	30	64	713.6	430.1	7.78	120	0.16 - 6500
512	622	50	+	70	10	120.8	430.1	7.78	260	0.04 - 1600
512	622	50	+	70	64	713.6	430.1	7.78	260	0.04 - 1600
512	622	20	+	30	10	120.8	430.1	7.78	120	0.04 - 1600
512	622	20	+	30	64	713.6	430.1	7.78	120	0.04 - 1600

Table 4: Fixed priority scheduler example.

6.2 WFQ scheduler

6.2.1 Determination of the scheduling tag

A WFQ scheduler [23, 36] uses a tag calculation of the form

$$V_i = \max(V(a_i), V_{i-1}) + \frac{ipps_i}{r_i}$$

with $V(a_i)$: global virtual time measure at the packet arrival time a_i, V_{i-1}: tag of the preceding packet of the same connection, $ipps_i, r_i$: packet size of current packet and the reserved rate of the connection respectively (the connection context).

- space: the global virtual time should be stored in a register or on-chip RAM. A global virtual time register is needed for every active scheduler in the node, i.e., up to (N-1) registers may be needed. Moreover, the context information is already saved in registers, since its value was determined during the IP router lookup. That is, only the tags of the preceding packet must be stored for every connection.
  - tag lookup implemented as table: $nc \cdot ts$

  - tag lookup implemented as p-trie, p pointers per node: $\frac{1}{p-1}(p^{\lceil \log_p nc \rceil} - 1) \cdot p \cdot addr_{ss} + nc \cdot ts$

- throughput:
  - table lookup: $\frac{r}{ipps_{min}} \cdot 2 \cdot ts$
  - trie implementation: $\frac{r}{ipps_{min}} \cdot ((\lceil \log_p nc \rceil \cdot addr_{ss} + ts) + ts)$ for reading and storing tag values

- calculations: a maximum calculation of the global virtual time and the tag of the preceding packet, an addition of the context information for every packet (max. $\frac{r}{ipps_{min}}$ times per second).
  - table lookup: one address offset calculation for reading out the previous tag. For storing the new tag, the calculated address can be reused.
  - trie implementation, p pointers per node: $\lceil \log_p nc \rceil$ address offset calculations for reading out the previous tag. For storing the new tag, the last jump address can be reused.

- delay for a packet:
  - table lookup: $3 \cdot clk^{-1} + 2 \cdot lat_{ss}$ with $g = ts$
  - trie implementation: $(\lceil \log_p nc \rceil + 2) \cdot clk^{-1} + (\lceil \log_p nc \rceil + 2) \cdot lat_{ss}$ with $g = max(addr_{ss}, ts, cp)$

6.2.2 Enqueue operation

The IP header must be inserted into a sorted priority queue according to its scheduling tag. The packets of connections with the same destination node on the ATM line use the same priority queue. A worst-case situation could arise if all nc connections were set up to the same destination node on the ATM line and therefore had to share a single priority queue. The size of the priority queue can be reduced by only looking at the head elements of the connection queues, that is, only the head elements must be sorted according to their priority tags. A heap organized binary tree [18] is used for the implementation of the priority queue. We have chosen this simple, mature data structure since we do not depend on quick list merge operations, since heaps can be analyzed well and have symmetric complexity for enqueue and dequeue operations and therefore still compete reasonably well with more sophisticated priority queue implementations [25, 17]. Moreover, since insert and delete operations occur frequently, a heap is preferred rather than a balanced tree implementation, since additional operations for maintaining a balanced data structure can cause noticeable overhead. However, a tree implementation is chosen instead of an array, since the number of elements to be stored cannot be statically determined. The binary tree consists of nodes with the following fields: two pointers to the left and right child and a value field. Globally, two values must be stored: the number of elements stored in the tree and a pointer to the root node. The binary coding of the number of nodes is used to traverse the tree in order to find the last inserted element or the new position for insertions. Traversing the tree, pointers to nodes along the path are temporarily saved (max. $\log_2 \lfloor nc \rfloor$ pointers) in registers in order to quickly find the nodes in the backward direction. Note that the exchange of two tree nodes is done by copying the value fields and not by adjusting pointers. The copying only requires two memory accesses. Adjusting pointers however would need up to five memory accesses.

- space:
  - lookup for destination node: $(N-1) \cdot addr_{ip}$
    for determining the corresponding priority queue with a simple table lookup.
  - heap organized binary tree (p-queue): $\sum_{d_i}(2 \cdot addr_{ip} + (ts + addr_{pl} + vis + con)) \cdot pn_{d_i}$

- throughput:
  - lookup: $\frac{r}{ipps_{min}} \cdot addr_{ip}$ for determining the priority queue.
  - heap organized binary tree:
    $\frac{r}{ipps_{min}}((\lfloor \log_2(nc+1) \rfloor + 1) \cdot addr_{ip} + \lfloor \log_2(nc+1) \rfloor \cdot 2 \cdot$
    $\cdot(ts + addr_{pl} + vis + con) + (ts + addr_{pl} + vis + con))$
    (find the insertion point, adjust pointer, worst case number of entry exchanges, write value field of new entry)
- calculations (per packet):
  - lookup: an address offset calculation for determining the priority queue.
  - heap organized binary tree: increment node counter for d_i's priority queue, allocate memory for new node, determine the position for insertion (almost no overhead, since a bit in the number of node decides, whether the left or right child node must be chosen), max. $\lfloor \log_2(nc+1) \rfloor$ comparisons of ts bit values
- delay: $(\lfloor \log_2(nc+1) \rfloor + 2) \cdot clk^{-1} + (3 \cdot \lfloor \log_2(nc+1) \rfloor + 2) \cdot lat_{ip}$ with $g = ts + addr_{pl} + vis + con$. Accesses may be divided into small address field accesses ($g = addr_{ip}$) and larger information field accesses. Then, the formula can be refined to $(\lfloor \log_2(nc+1) \rfloor + 2) \cdot clk^{-1} + (\lfloor \log_2(nc+1) \rfloor + 1) \cdot lat_{ip,addr} + (2 \cdot \lfloor \log_2(nc+1) \rfloor + 1) \cdot lat_{ip}$.

6.2.3 Dequeue operation

The schedulers of the priority queues all together must schedule packets at most $\frac{r}{ipps_{min}}$ times per second. A WFQ scheduler needs to find the packet with the highest priority in the priority queue what usually means finding the packet with the smallest scheduling tag.

- space: the space requirements are already considered at the enqueue operation.
- throughput (needed by the p-queue): $\frac{r}{ipps_{min}}((\lfloor \log_2 nc \rfloor + 1) \cdot addr_{ip} + \lfloor \log_2(nc-1) \rfloor \cdot 2 \cdot$
  $\cdot(ts + addr_{pl} + vis + con) + (ts + addr_{pl} + vis + con) + (addr_{pl} + vis + con))$
  (find the deletion point, reset pointer, worst case number of entry exchanges, write value field, read value field of top element)
- calculations (needed by p-queue): the packet entry with the smallest scheduling tag can be found quickly at the root of the binary tree. However, the entry at the last inserted position is exchanged with the root and enqueued again in the binary tree to maintain the heap organization. The last inserted position can be determined from the number of nodes in the tree. The needed operations for every packet are: $\lfloor \log_2 nc \rfloor$ comparisons of ts bit values, decrement node counter, deallocate memory of last inserted node.
- delay for the head of line packet of a flow, corresponds to total delay if the IP sources would be constant bit-rate sources:
  - caused by calculations and memory accesses:
    $(\lfloor \log_2 nc \rfloor + 1) \cdot clk^{-1} + (\lfloor \log_2 nc \rfloor + 2 \cdot \lfloor \log_2(nc-1) \rfloor + 3) \cdot lat_{ip}$. Again, accesses may be divided into small address field accesses and larger information field accesses. Then, the formula can be refined to $(\lfloor \log_2 nc \rfloor + 1) \cdot clk^{-1} + (\lfloor \log_2 nc \rfloor + 1) \cdot lat_{ip,addr} + (2 \cdot \lfloor \log_2(nc-1) \rfloor + 2) \cdot lat_{ip}$.

– scheduling algorithm characteristics: $\frac{ipps_i}{r_i} + \frac{ipps_{max}}{r}$. The term $\frac{ipps_i}{r_i}$ results from the guaranteed transport delay. The second term considers the situation where a maximum-sized packet has been chosen for transmission before the arrival of the current packet.

6.2.4 Context information update

Maintaining and updating the global virtual time V of the corresponding emulated fluid fair queuing system basically needs four registers per scheduler (i.e. active destination node): a register for storing V, a register for storing the sum of the reserved rates ($\sum_{B(t)} r_i$) of all backlogged connections, a register for storing the system time of the last virtual clock update, and a register for storing the time at which V must be updated next because a packet has been served in the fluid system. An update of V is performed as follows: $V(t+dt) := V(t) + \frac{dt}{\sum_{B(t)} r_i}$. $B(t)$ is the set of backlogged connections in the fluid system at time t. The time for the next update of V if no more packets arrive is given by $next(t) = t + (V_{i,min} - V(t)) \cdot \sum_{B(t)} r_i$. Note that the scheduler needs its own priority queue for the emulation of the fluid system since the contents of the fluid system priority queue and the WFQ priority queue may differ. Moreover, only average bounds are determined since they are calculated on a per packet basis. In the worst-case all virtually backlogged connections in the fluid system may be released at the same point of time and generate considerable bursts of events.

- space:
  - counter lookup: $nc \cdot cp$. Lookup in a vector of counters which states how many packets are still in the corresponding packet queue of the connection.
  - priority queue for determining $V_{i,min}$ in the fluid system (heap organized binary tree): $\sum_{d_i}(2 \cdot addr_{ss} + ts) \cdot pn_{d_i}$
- throughput:
  - counters: $4 \cdot \frac{r}{ipps_{min}} \cdot cp$ (enqueue and dequeue).
    It is assumed that at least one byte must be read or written.
  - p-queue enqueue and dequeue operations:
    $2 * \frac{r}{ipps_{min}}((\lfloor \log_2(nc+1) \rfloor + 1) \cdot addr_{ss} + \lfloor \log_2(nc+1) \rfloor \cdot 2 \cdot ts + ts)$
- calculations:
  - after each calculation of $next(t)$ a timer is set with the difference of $next(t)$ and the current system time.
  - enqueue (packet arrival): in the worst-case, a (virtual) connection queue becomes backlogged which was empty before. In order to determine the backlog state, an address offset calculation is needed. Moreover, the backlog counter must be incremented. Then, the sum register might be updated by the amount of the reserved rate of that connection if the queue was empty before. An addition operation is needed for that. Moreover, the ratio register must be recalculated with a division. V can be updated by another addition. Last, the calculation of $next(t)$ needs a subtraction, a multiplication, and an addition.
  - dequeue (in the emulated fluid system): after the dequeue operation, a (virtual) connection queue may no longer be backlogged. The backlog state must be adjusted accordingly. The backlog counter must be updated and therefore an address offset calculation and a decrement operation are required. The registers must be updated accordingly, that means, two subtractions, a division, a multiplication, and two additions are needed in the worst case.

– p-queue operations: $2 \cdot (\lfloor \log_2(nc+1) \rfloor + 1)$
(comparisons of ts values and increment/decrement of the nodes counter)

- delay per packet: $(16 + 2 \cdot (\lfloor \log_2(nc+1) \rfloor + 1)) \cdot clk^{-1} + (4 + 2 \cdot (3 \cdot \lfloor \log_2(nc+1) \rfloor + 2)) \cdot lat_{ss}$

6.2.5 Example

See Tab. 5. The additional delay (besides the transport delay) introduced by the characteristics of the scheduling algorithm ($\frac{ipps_{max}}{r}$) is about 80 μs for a line rate of 155 Mbit and has not been considered in the column *del*. Again, two access types have been distinguished. Short accesses for traversing the tree structure of the heap only require a $lat_{ip,addr}$ delay. Long transfers which access the value fields of the tree nodes need a lat_{ip} delay. Two packets per connection are queued in the system on the average.

6.3 WFQ approximation

Connections are grouped according to their service intervals, as suggested in [29], which are defined by $\Phi_i := \frac{ipps_i}{r_i}$. $ipps_i$ is the size of the packet at the head of the FIFO queue of connection i. In order to restrict the number of groups, the set of all possible service intervals must be reduced to a bounded set accepting a certain penalty in terms of delay and fairness. The range of possible service intervals is uniformly exponentially spanned by a set of G groups with a granularity Δ: $\Phi_{max} = \Delta^{G-1}\Phi_{min}$. The corresponding group g of a packet is therefore determined by $g = \lfloor \log_\Delta \frac{\Phi_i}{\Phi_{min}} \rfloor + 1$. Thus, when a packet of a connection has been served, the connection might belong to another service interval group. However, using this grouping scheme and accepting a certain delay penalty, sorting the heads of the connection queues within a group according to scheduling tags can be easily achieved by a simple linked list and two additional pointers, see Fig. 4. Moreover, the scheduler only needs to look at the scheduling tags of one packet of each group (the gray elements in Fig. 4), not at the tags of all packets in the system. The scheduling tags within a service interval group can span the range $\frac{ipps_{max}}{ipps_{min}}\Phi_g$ which also is

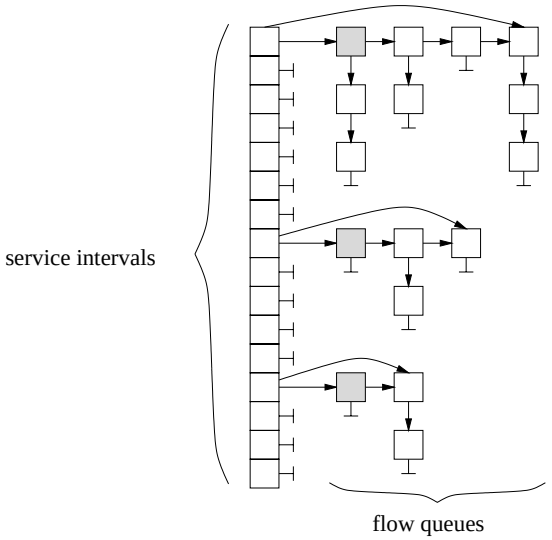

Figure 4: Sorting data structure.

the maximum delay penalty introduced by this grouping scheme. This value can be decreased by further

Fixed model parameters

$ipps_{min}$	40 byte	minimal packet size
vis	28 bit	VPI/VCI field size
con	14 bit	flow context information
$addr_{pl}$	22 bit	payload addresses
$addr_{ip}$	22 bit	packet header addresses
clk	200 MHz	CPU clock
r_{min}	128 Kbit	minimal supported rate of a flow
cp	16 bit	backlog counter precision
ts	32 bit	scheduling tag size

Model parameters						*WFQ, average case*							
N	r	nc	lat_{ss}	$lat_{ip,addr}$+lat_{ip}		sp_{ip}	sp_{ss}	th_{ip}	th_{ss}	ops_{ip}	ops_{ss}	del_{ip}	del_{ss}
	[Mbit/s]		[ns]	[ns]		[Kbit]	[Kbit]	[Mbit/s]	[Mbit/s]	[10^6]	[10^6]	[μs]	[μs]
256	155	2^6	50	50	+ 70	23.0	12.5	1246.3	600.5	8.2	16.0	2.53	1.57
256	155	2^6	50	20	+ 30	23.0	12.5	1246.3	600.5	8.2	16.0	1.11	1.57
256	155	2^6	20	50	+ 70	23.0	12.5	1246.3	600.5	8.2	16.0	2.53	0.73
256	155	2^6	20	20	+ 30	23.0	12.5	1246.3	600.5	8.2	16.0	1.11	0.73
256	155	2^{13}	50	50	+ 70	2245.5	1600	2630.3	1098	15.0	22.8	5.26	2.69
256	155	2^{13}	50	20	+ 30	2245.5	1600	2630.3	1098	15.0	22.8	2.30	2.69
256	155	2^{13}	20	50	+ 70	2245.5	1600	2630.3	1098	15.0	22.8	5.26	1.22
256	155	2^{13}	20	20	+ 30	2245.5	1600	2630.3	1098	15.0	22.8	2.30	1.22
256	622	2^6	50	50	+ 70	23.0	12.5	5001.3	2410	33.0	64.1	2.53	1.57
256	622	2^6	50	20	+ 30	23.0	12.5	5001.3	2410	33.0	64.1	1.11	1.57
256	622	2^6	20	50	+ 70	23.0	12.5	5001.3	2410	33.0	64.1	2.53	0.73
256	622	2^6	20	20	+ 30	23.0	12.5	5001.3	2410	33.0	64.1	1.11	0.73
256	622	2^{13}	50	50	+ 70	2245.5	1600	10555	4404	60.3	91.4	5.26	2.69
256	622	2^{13}	50	20	+ 30	2245.5	1600	10555	4404	60.3	91.4	2.30	2.69
256	622	2^{13}	20	50	+ 70	2245.5	1600	10555	4404	60.3	91.4	5.26	1.22
256	622	2^{13}	20	20	+ 30	2245.5	1600	10555	4404	60.3	91.4	2.30	1.22
512	155	2^6	50	50	+ 70	28.5	12.5	1246.3	600.5	8.2	16.0	2.53	1.57
512	155	2^6	50	20	+ 30	28.5	12.5	1246.3	600.5	8.2	16.0	1.11	1.57
512	155	2^6	20	50	+ 70	28.5	12.5	1246.3	600.5	8.2	16.0	2.53	0.73
512	155	2^6	20	20	+ 30	28.5	12.5	1246.3	600.5	8.2	16.0	1.11	0.73
512	155	2^{13}	50	50	+ 70	2251.0	1600	2630.3	1098	15.0	22.8	5.26	2.69
512	155	2^{13}	50	20	+ 30	2251.0	1600	2630.3	1098	15.0	22.8	2.30	2.69
512	155	2^{13}	20	50	+ 70	2251.0	1600	2630.3	1098	15.0	22.8	5.26	1.22
512	155	2^{13}	20	20	+ 30	2251.0	1600	2630.3	1098	15.0	22.8	2.30	1.22
512	622	2^6	50	50	+ 70	28.5	12.5	5001.3	2410	33.0	64.1	2.53	1.57
512	622	2^6	50	20	+ 30	28.5	12.5	5001.3	2410	33.0	64.1	1.11	1.57
512	622	2^6	20	50	+ 70	28.5	12.5	5001.3	2410	33.0	64.1	2.53	0.73
512	622	2^6	20	20	+ 30	28.5	12.5	5001.3	2410	33.0	64.1	1.11	0.73
512	622	2^{13}	50	50	+ 70	2251.0	1600	10555	4404	60.3	91.4	5.26	2.69
512	622	2^{13}	50	20	+ 30	2251.0	1600	10555	4404	60.3	91.4	2.30	2.69
512	622	2^{13}	20	50	+ 70	2251.0	1600	10555	4404	60.3	91.4	5.26	1.22
512	622	2^{13}	20	20	+ 30	2251.0	1600	10555	4404	60.3	91.4	2.30	1.22

Table 5: WFQ scheduler example.

subdividing the range of scheduling tags within a group linearly into x subgroups. The corresponding

subgroup can be determined by another division operation. A priority queue per group keeps track of the head elements of the subgroups.

6.3.1 Scheduling tag calculation

The described data structure can be applied for a variety of WFQ variants.

6.3.2 Enqueue operation

Using the connection identifier (VPI/VCI pair) a lookup is performed which determines the position of the end of the connection queue. If the connection queue is empty, the packet is added at the end of the linked list of the corresponding service interval group/subgroup to be the head element in its connection queue. The lookup vector must be adjusted to point to the new end of the corresponding connection queue as well as the pointer which directs to the end of the linked list of a service interval group. Moreover, the packet may be enqueued in the group's priority queue and the scheduler's priority queue in the worst-case. These transfers have been considered in the dequeue operation.

- space:
  - lookup vector (destination node): $(N-1) \cdot addr_{ip}$
    for determining the responsible scheduler
  - lookup vector (end of flow queue): $nc \cdot addr_{ip}$
  - service interval class lookup: $2 \cdot addr_{ip} \cdot nsic \cdot (N-1)$
  - list and queue elements: $\sum_{d_i}(2 \cdot addr_{ip} + (addr_{pl} + vis + con + ts)) \cdot max(pn_{d_i} - nsic - nsic \cdot x, 0)$
  - priority queue elements (scheduler), heap organized array [18]:
    $(N-1) \cdot nsic \cdot (ts + addr_{ip} + vis + con + \lceil \log_2 nsic \rceil)$
  - subgroup priority queues (x entries per group): $(N-1) \cdot nsic \cdot x \cdot (ts + addr_{ip} + vis + con + \lceil \log_2 x \rceil)$
- throughput:
  - determining the scheduler: $\frac{r}{ipps_{min}} \cdot addr_{ip}$
  - queuing: $\frac{r}{ipps_{min}}(2 \cdot addr_{ip} + (addr_{pl} + vis + con + ts) + 2 \cdot addr_{ip})$
    (two lookups, writing of the value field, pointer updates)
- calculations for every packet: an address offset calculation for determining the responsible scheduler, an address offset calculation for determining the state of the corresponding connection queue, calculation of the matching service interval class (a division, a *log* operation (counted as 10 calculations), a floor operation and an increment) and subgroup (a division), an address offset calculation for determining the end of the linked list if connection queue empty, allocate memory for new entry, update pointers in class lookup vector and at the end of the list.
- delay per packet: $18 \cdot clk^{-1} + 6 \cdot lat_{ip}$ with $g = max(addr_{ip}, addr_{pl} + vis + con + ts)$. Accesses may be divided into small address field accesses ($g = addr_{ip}$) and larger information field accesses. Then, the formula can be refined to $18 \cdot clk^{-1} + 5 \cdot lat_{ip,addr} + lat_{ip}$.

6.3.3 Dequeue operation

The scheduler has to choose the packet with the smallest tag among all minimal tags of each service interval group. The number of groups (and minimal tags) is independent of the number of connections which are supported by the scheduler. After a packet has been dequeued, the new head of the corresponding connection queue must be checked whether the queue has to be inserted into another service interval group. At least, the two pointers which direct to the beginning and the end of the linked list of connections within a group must be adjusted and the new head of the class' queue has to be copied to the priority queue of the scheduler. Moreover, a further packet may be enqueued in the group's priority queue.

- space: already considered at the enqueue operation
- throughput: $\frac{r}{ipps_{min}}((2 \cdot addr_{ip} + \lceil \log_2 nsic \rceil + \lceil \log_2 x \rceil) + (\lfloor \log_2(nsic-1) \rfloor + \lfloor \log_2(x-1) \rfloor) \cdot 2 \cdot (ts + addr_{pl} + vis + con) + 4 \cdot (ts + addr_{pl} + vis + con) + 2 \cdot addr_{ip} + 3 \cdot addr_{ip})$
  (read out of top elements of both priority queues, worst-case number of entry exchanges in the heap arrays, read new elements from the corresponding service interval class queue and subclass list, store it in the heap arrays, update pointer of the class, reinsert connection queue if necessary)
- calculations per packet: $\lfloor \log_2(nsic-1) \rfloor + \lfloor \log_2(x-1) \rfloor$ address offset calculations for the heap accesses, an address offset calculation for determining the current service interval class, deallocate memory for one entry, calculate the service interval class and the subgroup of the next entry (14 operations) as well as calculate an address offset if a reinsertion is necessary.
- delay per packet
  - caused by calculations and memory accesses: $(\lfloor \log_2(nsic-1) \rfloor + \lfloor \log_2(x-1) \rfloor + 17) \cdot clk^{-1} + (11 + 2 \cdot (\lfloor \log_2(nsic-1) \rfloor + \lfloor \log_2(x-1) \rfloor)) \cdot lat_{ip}$. Again, accesses may be divided into small address field accesses and larger information field accesses. Then, the formula can be refined to $(\lfloor \log_2(nsic-1) \rfloor + \lfloor \log_2(x-1) \rfloor + 17) \cdot clk^{-1} + 5 \cdot lat_{ip,addr} + (6 + 2 \cdot (\lfloor \log_2(nsic-1) \rfloor + \lfloor \log_2(x-1) \rfloor)) \cdot lat_{ip}$.
  - because of scheduling algorithm characteristics: The additional worst-case delay introduced by the characteristics of the grouping scheme increases the guaranteed transport delay (ideal service interval) of the current packet by a factor of $(1 + \frac{ipps_{max}}{ipps_{min} \cdot x})$.

6.3.4 Context information update

Dependent on the underlying scheduling algorithm, virtual service may be managed according to e.g. WFQ (subsection 6.2.4) or MD-SCFQ (subsection 6.4.4).

6.3.5 Example

See Tab. 6. Using our settings in this example, $nsic = 16$ service intervals result in a granularity of $\Delta = 2$. On the average, two packets per connection are queued in the system in this example. The additional delay for the delay bound of the scheduler introduced by the grouping scheme increases the ideal transport delay by a factor of $(1 + \frac{ipps_{max}}{ipps_{min} \cdot x})$ in the worst-case. Using $x = 32$ subclasses per service interval class, the approximation doubles the transport delay bound, for $x = 128$ the transport delay bound increases by 30 %. Compared with the WFQ example, the bounds for memory space, throughput and calculations have been reduced at the expense of the delay bound.

Fixed model parameters

$ipps_{max}$	1536 byte	maximal packet size
$ipps_{min}$	40 byte	minimal packet size
vis	28 bit	VPI/VCI field size
con	14 bit	flow context information
$addr_{pl/ip}$	22 bit	payload/packet header addresses
clk	200 MHz	CPU clock
r_{min}	128 Kbit	minimal supported rate of a flow
r_{max}	10^8 Mbit	maximal supported rate of a flow
ts	32 bit	scheduling tag size
$nsic$	16	number of service intervals

Model parameters						*WFQ approximation*			
N	r [Mbit/s]	nc	$lat_{ip,addr}$+lat_{ip} [ns]		x	sp_{ip} [Mbit]	th_{ip} [Mbit/s]	ops_{ip} [10^6]	del_{ip} [μs]
256	155	2^6	50	+ 70	32	7.75	968.7	20.3	1.9
256	155	2^6	50	+ 70	128	30.78	1147.0	21.3	2.1
256	155	2^6	20	+ 30	32	7.75	968.7	20.3	0.9
256	155	2^6	20	+ 30	128	30.78	1147.0	21.3	1.0
256	155	2^{13}	50	+ 70	32	10.04	968.7	20.3	1.9
256	155	2^{13}	50	+ 70	128	32.87	1147.0	21.3	2.1
256	155	2^{13}	20	+ 30	32	10.04	968.7	20.3	0.9
256	155	2^{13}	20	+ 30	128	32.87	1147.0	21.3	1.0
256	622	2^6	50	+ 70	32	7.75	3887.2	81.6	1.9
256	622	2^6	50	+ 70	128	30.78	4602.7	85.5	2.1
256	622	2^6	20	+ 30	32	7.75	3887.2	81.6	0.9
256	622	2^6	20	+ 30	128	30.78	4602.7	85.5	1.0
256	622	2^{13}	50	+ 70	32	10.04	3887.2	81.6	1.9
256	622	2^{13}	50	+ 70	128	32.87	4602.7	85.5	2.1
256	622	2^{13}	20	+ 30	32	10.04	3887.2	81.6	0.9
256	622	2^{13}	20	+ 30	128	32.87	4602.7	85.5	1.0
512	155	2^6	50	+ 70	32	15.53	968.7	20.3	1.9
512	155	2^6	50	+ 70	128	61.69	1147.0	21.3	2.1
512	155	2^6	20	+ 30	32	15.53	968.7	20.3	0.9
512	155	2^6	20	+ 30	128	61.69	1147.0	21.3	1.0
512	155	2^{13}	50	+ 70	32	17.82	968.7	20.3	1.9
512	155	2^{13}	50	+ 70	128	63.77	1147.0	21.3	2.1
512	155	2^{13}	20	+ 30	32	17.82	968.7	20.3	0.9
512	155	2^{13}	20	+ 30	128	63.77	1147.0	21.3	1.0
512	622	2^6	50	+ 70	32	15.53	3887.2	81.6	1.9
512	622	2^6	50	+ 70	128	61.69	4602.7	85.5	2.1
512	622	2^6	20	+ 30	32	15.53	3887.2	81.6	0.9
512	622	2^6	20	+ 30	128	61.69	4602.7	85.5	1.0
512	622	2^{13}	50	+ 70	32	17.82	3887.2	81.6	1.9
512	622	2^{13}	50	+ 70	128	63.77	4602.7	85.5	2.1
512	622	2^{13}	20	+ 30	32	17.82	3887.2	81.6	0.9
512	622	2^{13}	20	+ 30	128	63.77	4602.7	85.5	1.0

Table 6: WFQ approximation example.

6.4 MD-SCFQ

MD-SCFQ [4] is a further development of self-clocked fair queuing (SCFQ) [11]. SCFQ approximates the global virtual time of the fluid system in WFQ by the scheduling tag of the packet currently in service in the packet system. In other words, updates are only triggered by packet arrivals and departures in the packet system. SCFQ does not need to emulate a fluid system. However, this approximation leads to much weaker worst-case delay bounds. MD-SCFQ therefore approximates the global virtual time measure linearly between updates by a so-called system potential (applying the framework presented in [31]). Recalibrations of the global virtual time are performed at the end of every packet departure in the packet system. The system potential is updated at every packet arrival and departure. That is, updates and recalibrations are still triggered by the packet system, but the global virtual time of the fluid system is emulated more precisely than in SCFQ. MD-SCFQ achieves the same theoretical delay bounds based on algorithm characteristics as WFQ.

6.4.1 Scheduling tag calculation

The bounds for the calculation of the scheduling tags can be taken from the WFQ scheduler bounds in subsection 6.2. The main difference between WFQ and MD-SCFQ, the approximation of the global virtual time by a step-wise linear system potential, is considered in subsection 6.4.4.

6.4.2 Enqueue operation

The data structure needed for an implementation of a priority queue can be adopted from the WFQ implementation in subsection 6.2 or the WFQ approximation in subsection 6.3.

6.4.3 Dequeue operation

The requirements for the dequeue operation can be adopted from the preceding subsections 6.2.3 and 6.3.3 dependent on the chosen data structure. The delay bounds introduced by the characteristics of the scheduling algorithm are the same as the bounds derived for WFQ.

6.4.4 Context information update

The exact global virtual time measure $S^P(t)$ used in MD-SCFQ is calculated as follows: $S^P(t) = \frac{F_{B(t)} - L_{B(t)}}{r_{B(t)}}$ with $F_{B(t)} = \sum_{B(t)} V_i r_i$ weighted sum of the scheduling tags, $L_{B(t)} = \sum_{B(t)} ipps_i$ sum of lengths of the head of session packets, $r_{B(t)} = \sum_{B(t)} r_i$ cumulated service rate. The system potential $P(t)$, which is used as a piece-wise linear approximation of the global virtual time, increases linearly with time: $P(t+dt) = P(t)+dt$. A recalibration is performed at every packet departure: $P(t) = max(P(t-), S^P(t))$. Registers/ on-chip RAM are needed to hold $F_{B(t)}$, $L_{B(t)}$, $r_{B(t)}$, $P(t)$, and the system time of the last update of the system potential for every scheduler (active destination node).

- space: $nc \cdot cp$. Lookup in a vector of counters which states how many packets are still in the corresponding packet queue of the flow.
- throughput (caused by counters): $4 \cdot \frac{r}{ipps_{min}} \cdot cp$
  (enqueue and dequeue). It is assumed that at least one byte must be read or written.
- calculations:

– enqueue (packet arrival): in the worst-case, a connection queue becomes backlogged which was empty before. In order to determine the backlog state, an address offset calculation is needed and the corresponding backlog counter must be incremented. Moreover, $F_{B(t)}$, $L_{B(t)}$, $r_{B(t)}$, $P(t)$ might be updated by four additions, a subtraction, and a multiplication.

– dequeue (packet departure in the packet system): after the dequeue operation, the corresponding connection may no longer be backlogged. The backlog counter must be updated by an address offset calculation and a decrement operation. Moreover, $F_{B(t)}$, $L_{B(t)}$, $r_{B(t)}$, $P(t)$ might be updated by four additions, four subtractions, and a multiplication. The update of $S^P(t)$ needs a subtraction and an addition. Finally, the recalibration of $P(t)$ needs a max operation.

- delay per packet: $(8 + 14) \cdot clk^{-1} + 4 \cdot lat_{ss}$

6.4.5 Example

See Tab. 7. Due to the simplified calculation and update of the virtual service measure, the impact on

Fixed model parameters		
$ipps_{min}$	40 byte	minimal packet size
clk	200 MHz	CPU clock
cp	16 bit	backlog counter precision
ts	32 bit	scheduling tag size

Model parameters			*MD-SCFQ*			
r	nc	lat_{ss}	sp_{ss}	th_{ss}	ops_{ss}	del_{ss}
[Mbit/s]		[ns]	[Kbit]	[Mbit/s]	[10^6]	[ns]
155	2^6	50	3.0	59.1	10.7	425
155	2^6	20	3.0	59.1	10.7	245
155	2^{13}	50	384.0	59.1	10.7	425
155	2^{13}	20	384.0	59.1	10.7	245
622	2^6	50	3.0	237.3	42.8	425
622	2^6	20	3.0	237.3	42.8	245
622	2^{13}	50	384.0	237.3	42.8	425
622	2^{13}	20	384.0	237.3	42.8	245

Table 7: MD-SCFQ example.

the memory and CPU usage has been reduced compared with the WFQ system. Moreover, the values given in the MD-SCFQ examples are worst-case bounds, whereas the examples for WFQ show average bounds. In the WFQ case, updates are performed asynchronously to packet arrivals and departures of the real system according to events in an emulated fluid system.

7 IP payload - IP header split and storage

Specific parameters

$addr_{pl}$	$[bit]$	number of address bits needed for random accesses in the memory used for the payload storage
lat_{pl}	$[s]$	average memory access latency for payload RAM

7.1 Payload and address queue RAM

The address queue and the payload RAM should be implemented in the same memory area. Using two distinct memory chips would be a waste of resources since a full address queue implies an empty payload RAM and vice versa. The address queue is simply a FIFO organized linked list of pointers which direct to free memory areas of the size of an ATM cell payload field (48 byte).

- space required: $2 \cdot addr_{pl}$ for an address queue entry, $\lceil \frac{ipps}{cs-hs} \rceil \cdot (cs - hs)$ payload storage for a packet of the size *ipps*, two registers for the FIFO organization of the address queue (pointers to the first and the last element of the queue)
- throughput: $\frac{r}{cs} \cdot (3 \cdot addr_{pl} + cs - hs)$ for payload storage and updating pointers, $\frac{r}{cs} \cdot (2 \cdot addr_{pl} + cs - hs)$ for reading out payload and generating corresponding entries in the address queue.
- calculations: $\lceil \frac{ipps}{cs-hs} \rceil$ payload RAM segments of the size $cs - hs$ must be allocated in order to store an IP packet. Therefore, the number of segments must be calculated by a division at most $r \cdot \frac{\frac{cs-hs}{cs}}{ipps_{min}}$ times. The same number of address queue entries must be deallocated. No additional operations are required for the storage of the payload. $\lceil \frac{ipps}{cs-hs} \rceil$ address entries must be allocated for the address queue when an IP payload entry has been read out.
- delay per packet: $(4 \cdot \lceil \frac{ipps}{cs-hs} \rceil + 1) clk^{-1} + \lceil \frac{ipps}{cs-hs} \rceil \cdot 7 \cdot lat_{pl}$ with $g = max(addr_{pl}, cs - hs)$. Since the granularity needed for accessing payload addresses and payload fields differs by more than one order of magnitude, one may think of splitting the payload transfer in smaller bursts or breaking a long burst for address accesses. The delay per packet is then: $(4 \cdot \lceil \frac{ipps}{cs-hs} \rceil + 1) clk^{-1} + \lceil \frac{ipps}{cs-hs} \rceil \cdot (5 \cdot lat_{pl,addr_{pl}} + 2 \cdot lat_{pl,cs-hs})$.

7.2 Example

See Tab. 8. The memory space required depends on the amount of UBR traffic and the characteristics

Fixed model parameters		
cs	424 bit	cell size
hs	40 bit	cell header size
$ipps_{min}$	40 byte	minimal packet size
$addr_{pl}$	22 bit	payload addresses
clk	200 MHz	CPU clock

Model parameters			*packet split and storage*		
r	$lat_{pl,addr_{pl}}$	$lat_{pl,cs-hs}$	th	ops	del
[Mbit/s]	[ns]	[ns]	[Mbit/s]	[10^6]	[ns]
155	50	160	306.1	1.9	18880
155	20	85	306.1	1.9	9280
622	50	160	1228.3	7.6	18880
622	20	85	1228.3	7.6	9280

Table 8: IP packet split and storage example.

of the IP flows such as burst lengths. For instance, supporting up to 2^{13} IP flows with an average of two maximum sized packets queued in the system requires 24.1 MByte of payload storage.

8 VPI/VCI lookup for cell removal from line

Specific parameters

$addr_{vis}$	$[bit]$	number of address bits needed for random accesses in the memory used for the lookup table storage
lat_{vis}	$[s]$	average memory access latency for lookup table RAM
ne		number of connections ending in the current ATM node
n_{ubr}		number of UBR connections passing the current ATM node

8.1 VPI/VCI lookup

The lookup is needed in order to filter the packets which have to leave the ATM line at the node. This is the case if the packet's destination node is reached or the packet belongs to an UBR connection.

8.1.1 trie implementation

Complete trie with p pointers per node

- space: the amount of $\frac{1}{p-1}(p^{\lceil \log_p 2^{vis} \rceil} - 1) \cdot p \cdot addr_{vis} + 2^{vis} \cdot \lceil \log_2 N \rceil$ is needed for a fully populated trie. However, since only nc connections must be supported, the worst-case can be bounded by $\frac{1}{p-1}(p^{\lfloor \log_p nc \rfloor + 1} - 1) \cdot p \cdot addr_{vis} + nc \cdot (\lceil \log_p 2^{vis} \rceil - \lceil \log_p nc \rceil) \cdot p \cdot addr_{vis} + nc \cdot \lceil \log_2 N \rceil$.
- throughput: $\frac{r}{cs}(\lceil \log_p 2^{vis} \rceil \cdot addr_{vis} + \lceil \log_2 N \rceil)$
- calculations per cell: $\lceil \log_p 2^{vis} \rceil$ address offset calculations, one comparison of two $\lceil \log_2 N \rceil$ values
- delay per cell: $(\lceil \log_p 2^{vis} \rceil + 1) \cdot (clk^{-1} + lat_{vis})$ with $g = max(addr_{vis}, \lceil \log_2 N \rceil)$

8.1.2 Patricia tree [21] implementation

Only entries corresponding to UBR connections or connections ending in the current node can be found in the tree. Note that we consider also the average case using Patricia trees since in the worst-case Patricia trees are as bad as general binary trees. However, in the average case, a Patricia tree allows to skip unpopulated levels of the tree. In order to calculate the worst-case, replace the term $\lceil log_2(ne+n_{ubr}) \rceil$ by vis in the following formula.

- space: $(ne + n_{ubr}) \cdot (\lceil \log_2 N \rceil + 2 \cdot addr_{vis} + \lceil \log_2 \lceil \log_2 N \rceil \rceil)$
- throughput, average case: $\frac{r}{cs}(\lceil log_2(ne + n_{ubr}) \rceil \cdot (addr_{vis} + \lceil \log_2 \lceil \log_2 N \rceil \rceil) + \lceil \log_2 N \rceil)$
- calculations: in every step through the levels of the tree, a bit of the VPI/VCI pair decides whether the left or right child node must be chosen. When the destination node in the tree is reached, a comparison of two $\lceil \log_2 N \rceil$ bit values must be performed which decides whether the ATM cell must be taken from the ATM line.
- delay per cell, average case: $(\lceil log_2(ne + n_{ubr}) \rceil + 1) \cdot (clk^{-1} + lat_{vis})$
  with $g = max(addr_{vis} + \lceil \log_2 \lceil \log_2 N \rceil \rceil, \lceil \log_2 N \rceil)$

8.2 Example

See Tab. 9.

Fixed model parameters

cs	424 bit	cell size
vis	28 bit	VPI/VCI field size
$addr_{vis}$	22 bit	addresses for cell removal lookup
clk	200 MHz	CPU clock

Model parameters				2^8*-trie implementation*				*Patricia tree, average case*			
r	N	$ne + n_{ubr}$	lat_{vis}	sp	th	ops	del	sp	th	ops	del
[Mbit/s]			[ns]	[Mbit]	[Mbit/s]	[10^6]	[ns]	[Kbit]	[Mbit/s]	[10^6]	[ns]
155	256	2^{13}	50	44.4	33.5	2.56	275	440	116.1	5.1	770
155	256	2^{13}	20	44.4	33.5	2.56	125	440	116.1	5.1	350
155	256	2^6	50	0.09	33.5	2.56	275	3.44	55.1	2.6	385
155	256	2^6	20	0.09	33.5	2.56	125	3.44	55.1	2.6	175
155	512	2^{13}	50	44.4	33.8	2.56	275	456	121.8	5.1	770
155	512	2^{13}	20	44.4	33.8	2.56	125	456	121.8	5.1	350
155	512	2^6	50	0.09	33.8	2.56	275	3.56	57.5	2.6	385
155	512	2^6	20	0.09	33.8	2.56	125	3.56	57.5	2.6	175
622	256	2^{13}	50	44.4	134.3	10.27	275	440	465.9	20.5	770
622	256	2^{13}	20	44.4	134.3	10.27	125	440	465.9	20.5	350
622	256	2^6	50	0.09	134.3	10.27	275	3.44	221.0	10.3	385
622	256	2^6	20	0.09	134.3	10.27	125	3.44	221.0	10.3	175
622	512	2^{13}	50	44.4	135.7	10.27	275	456	485.5	20.5	770
622	512	2^{13}	20	44.4	135.7	10.27	125	456	485.5	20.5	350
622	512	2^6	50	0.09	135.7	10.27	275	3.56	230.8	10.3	385
622	512	2^6	20	0.09	135.7	10.27	125	3.56	230.8	10.3	175

Table 9: Cell removal example.

9 ATM payload-header split

This step is needed when ATM cells belonging to UBR traffic must be taken from the ATM line and reordered into the corresponding line access queue before the wheel. The requirements of this block are already considered by the worst case bounds of the payload and header queues.

Specific parameters

$addr_{pl}$	$[bit]$	number of address bits needed for random accesses in the memory used for the payload storage
lat_{pl}	$[s]$	average memory access latency for payload RAM
lat_{hd}	$[s]$	average memory access latency for destination queue RAM

9.1 ATM cell split and storage

- space: each UBR ATM cell needs one entry in the payload RAM ($cs - hs$ bits). Finally, the header must be stored in the UBR traffic queue before the wheel (($hs + addr_{pl}$ bits).
- throughput: in the worst case (only UBR traffic from the line) $\frac{r}{cs}(3 \cdot addr_{pl} + (cs - hs))$ is the throughput required from the payload RAM (read out of an address queue entry and payload storage). Moreover, $\frac{r}{cs}(hs + addr_{pl})$ throughput needed from the UBR destination queue RAM.

- calculations: memory handling: for every UBR cell, an entry in the payload RAM must be allocated by freeing an entry of the address queue. In addition, an entry in the UBR destination queue must be allocated.
- delay per cell: $max(lat_{pl}, lat_{hd})$ with $g_{pl} = max(addr_{pl}, cs - hs)$ and $g_{hd} = addr_{pl} + hs$

10 Reassembly

Specific parameters

$addr_{ra}$	$[bit]$	number of address bits needed for random accesses in the memory used for the ATM cell storage before the reassembly stage
lat_{ra}	$[s]$	average memory access latency for reassembly RAM
ne		number of connections ending in the current ATM node

10.1 Reassembly queues

There is a FIFO queue for every connection ending at this ATM node. The queues buffer ATM cells until an IP packet can be reassembled. A Patricia tree [21] data structure is used for determining the FIFO queue belonging to the current cell. Pointers to the head element and the last element of the corresponding queue are stored in a tree node. Note that we consider the average case using Patricia trees. In order to calculate the worst-case, replace the term $\lceil log_2 ne \rceil$ by vis in the following formula.

- space: $ne \cdot (4 \cdot addr_{ra} + \lceil \log_2 \lceil \log_2 ne \rceil \rceil + \lceil \frac{ipps_{max}}{cs-hs} \rceil \cdot cs)$
- throughput, average case: $\frac{r}{cs}(\lceil \log_2 ne \rceil \cdot (addr_{ra} + \lceil \log_2 \lceil \log_2 ne \rceil \rceil) + 4 \cdot addr_{ra} + 2 \cdot cs)$ (steps needed for traversing the tree, accessing the FIFO queue and transferring the cell content (write and read)).
- calculations, average case: allocation or deallocation of a cell entry for a FIFO queue, e.g. with the help of an additional address queue. Moreover, up to $\lceil \log_2 ne \rceil$ times per ATM cell insertion, an arbitrary bit of the VPI/VCI pair determines whether the left or the right child node will be accessed next.
- delay per packet, average case: $\lceil \frac{ipps_{max}}{cs-hs} \rceil \cdot ((\lceil \log_2 ne \rceil + 2) \cdot clk^{-1} + (\lceil \log_2 ne \rceil + 6) \cdot lat_{ra})$ with $g = max(cs, addr_{ra} + \lceil \log_2 \lceil \log_2 ne \rceil \rceil)$. Since the granularity needed for accessing addresses and complete cells differs by more than one order of magnitude, one may think of splitting the cell transfer in smaller bursts or breaking a long burst for address accesses. The delay per cell is then: $\lceil \frac{ipps_{max}}{cs-hs} \rceil \cdot ((\lceil \log_2 ne \rceil + 2) \cdot clk^{-1} + (\lceil \log_2 ne \rceil + 4) \cdot lat_{ra,addr} + 2 \cdot lat_{ra,cs})$

10.2 Example

See Tab. 10.

Fixed model parameters		
cs	424 bit	cell size
hs	40 bit	cell header size
$ipps_{max}$	1536 byte	maximal packet size
$addr_{ra}$	22 bit	reassembly RAM addresses
clk	200 MHz	CPU clock

Model parameters				*Patricia trie, average case*						
r	ne	$lat_{ra,addr}$	$lat_{ra,cs}$	sp_{hs}	sp_{cs}	th_{hs}	th_{cs}	ops	del_{hs}	del_{cs}
[Mbit/s]		[ns]	[ns]	[Kbit]	[Mbit]	[Mbit/s]	[Mbit/s]	[10^6]	[μs]	[μs]
155	2^{13}	50	180	736	106.0	148.5	295.6	5.5	29.6	11.5
155	2^{13}	50	96	736	106.0	148.5	295.6	5.5	29.6	6.1
155	2^{13}	20	180	736	106.0	148.5	295.6	5.5	13.3	11.5
155	2^{13}	20	96	736	106.0	148.5	295.6	5.5	13.3	6.1
155	2^6	50	180	5.7	0.8	83.0	295.6	2.9	17.3	11.5
155	2^6	50	96	5.7	0.8	83.0	295.6	2.9	17.3	6.1
155	2^6	20	180	5.7	0.8	83.0	295.6	2.9	7.7	11.5
155	2^6	20	96	5.7	0.8	83.0	295.6	2.9	7.7	6.1
622	2^{13}	50	180	736	106.0	596.0	1186.4	22.0	29.6	11.5
622	2^{13}	50	96	736	106.0	596.0	1186.4	22.0	29.6	6.1
622	2^{13}	20	180	736	106.0	596.0	1186.4	22.0	13.3	11.5
622	2^{13}	20	96	736	106.0	596.0	1186.4	22.0	13.3	6.1
622	2^6	50	180	5.7	0.8	158.1	1186.4	11.7	17.3	11.5
622	2^6	50	96	5.7	0.8	158.1	1186.4	11.7	17.3	6.1
622	2^6	20	180	5.7	0.8	158.1	1186.4	11.7	7.7	11.5
622	2^6	20	96	5.7	0.8	158.1	1186.4	11.7	7.7	6.1

Table 10: Reassembly example.

11 Enhanced IP router

Specific parameters

nr		number of routes in the routing table
$ipas$	[bit]	IP source + destination address field size
$addr_{ipr}$	[bit]	number of address bits needed for random accesses in the memory used for the IP router
lat_{ipr}	[s]	average memory access latency for router RAM

11.1 IP router/ forwarding stage

The algorithms investigated in this section assume a backbone router environment, i.e. a routing table with several 10000 entries. Enterprise routers with less than 1000 routing table entries probably rather use hashing-based algorithms. The algorithms are based on Patricia trie data structures [21] and compressed prefix tries [34] respectively. On the one hand, Patricia tries have been chosen for comparison since many variations of the basic data structure are used in software implementations of routing stages by Unix kernels [28]. On the other hand, looking at state of the art routing mechanisms, we wanted to choose a suitable trade-off between the number of required memory accesses and the memory space. The work presented by Gupta et al. in [14] minimizes the number of memory accesses. In the worst-case, only two accesses are needed. This result however is obtained at the expense of memory space

by precomputing an up to 24 bit wide prefix table. More than 30 Mbytes of RAM are then needed. Contrary to that, Degermark et al. [8] use a very dense data structure of only 160 Kbytes for storing the routing table in order to use on-chip caches for that purpose. However, more than ten memory accesses may be needed in order to find the matching routing entry. Level compressed (LC) tries applied to routing tables [22] show two pleasant properties. On the one hand, weakly covered regions of a binary prefix trie can be compressed using skip values along the branches. On the other hand, completely occupied subtries can be converted into efficient array structures. However, in order to estimate the required memory accesses and memory space further knowledge about the address prefix distribution is required. Another very similar approach using variable length subtables is presented in [15]. Lampson et al. [19] make use of cache lines in order to store efficient representations of subtries. Again, further information about the address prefix distribution is required in order to derive worst-case bounds for the memory usage. We have therefore chosen the data structure described by Tzeng et al. in [34] for the estimation of worst-case bounds. The bounds only depend on the number of routing prefixes typically stored in today's biggest routing tables and not on the distribution of the prefixes. Nevertheless, the obtained bounds are competitive compared with the bounds of the other cited papers which need real-world routing tables for worst-case estimates. Moreover, a prefix trie compression scheme is introduced which allows the efficient storage of a subtrie in fixed-sized memory segments. Waldvogel et al. [35] also determine worst-case bounds independent on the distribution of address prefixes by performing a binary search on prefix lengths. However, the underlying data structure is based on perfect hashes. The resulting bounds for IPv4 lookups are therefore worse than the bounds presented by Tzeng et al. Waldvogel's scheme however better scales for IPv6 addresses. In this report, only IPv4 lookups are considered.

11.1.1 Patricia tree implementation

A Patricia tree [21] is applied to routing lookups. The tree is searched for an IP source and destination address pair and gives the destination node ID, a VPI/VCI pair, a new IP address, and context information back. Note that we consider the average case using Patricia trees. In order to calculate the worst-case, replace the term $\lceil \log_2 max(nc, nr) \rceil$ by $ipas$ in the following formula. The routing / forwarding functionality is implemented by a full match search in our configuration. Forwarding filter rules must therefore adjust all corresponding entries in the tree.

- space: $max(nc, nr) \cdot ((vis + con + \frac{ipas}{2} + \lceil \log_2 N \rceil) + 2 \cdot addr_{ipr} + \lceil \log_2 ipas \rceil)$
- throughput, average case:
  $\frac{r}{ipps_{min}}(\lceil \log_2 max(nc, nr) \rceil \cdot (addr_{ipr} + \lceil \log_2 ipas \rceil) + (vis + con + \frac{ipas}{2} + \lceil \log_2 N \rceil))$
- calculations: Traversing the tree an arbitrary bit of the IP source/destination address pair determines at each node whether the left or the right child node must be accessed next.
- delay per packet, average case: $\lceil \log_2 max(nc, nr) \rceil \cdot clk^{-1} + (\lceil \log_2 max(nc, nr) \rceil + 1) \cdot lat_{ipr}$ with $g = max(addr_{ipr} + \lceil \log_2 ipas \rceil, vis + con + \frac{ipas}{2} + \lceil \log_2 N \rceil)$. Accesses may be divided into small address field accesses ($g = addr_{ipr} + \lceil \log_2 ipas \rceil$) and larger information field accesses. Then, the formula can be refined to $\lceil \log_2 max(nc, nr) \rceil \cdot clk^{-1} + \lceil \log_2 max(nc, nr) \rceil \cdot lat_{ipr,addr} + lat_{ipr}$.

11.1.2 Compressed prefix trie implementation

This implementation uses compressed prefix tries combined with a precomputed prefix table as described in [34]. These tries are preferred to level-compressed tries ([22]), multiway binary search data structures

([19]), and hashing-based implementations ([35]) in order to determine worst-case bounds independent on the distribution of routing prefixes. Compressed tries are only investigated for IPv4 addresses. Moreover, the cited algorithms only deal with routing. They do not provide packet filter/classification functionality. That is, in our case, the trie is searched for the longest prefix match of the destination IP address and returns the destination node ID, a VPI/VCI pair, and optionally a new IP address. Context information such as the reserved rate of a packet flow must be returned by an additional filter stage.

Tzeng et al. use a 17 bit precomputed prefix table. A table entry points to a binary prefix trie which is partitioned into subtries of a minimal depth 5 and a maximal number of 31 nodes (the depth corresponds to the number of levels). Each node of such a subtrie has been encoded by three bit. A subtrie is fetched by a single memory access. Therefore, at most five memory accesses are necessary in order to access a routing table entry: one access for the precomputed table, at most three accesses for the subtries, and one final access of the routing information (destination node ID etc.). One can easily construct another scheme based on compressed tries which uses a 16 bit precomputed prefix table and at most two levels of subtries of depth 8. This scheme would save a memory access. However, the memory space requirements would grow by a factor of about four since a compressed trie of depth 8 needs more than 100 Byte of storage.

- space: since current routing tables contain considerably less than 2^{17} prefix entries, an entry of the 17 bit prefix table only addresses a single prefix through three subtrie nodes in the worst-case. Note, that the assumption of traversing three subtries per prefix is a weak bound. In that particular case, a single subtrie would suffice. Tzeng et al. need 10 Byte to store a subtrie together with two pointers which are needed to access routing information and further subtries.
  $2^{17} \cdot addr_{ipr} + nr \cdot (3 \cdot 10\ byte + vis + \frac{ipas}{2} + \lceil \log_2 N \rceil)$ (table, subtries, and routing information)
- throughput: $\frac{r}{ipps_{min}}(addr_{ipr} + 3 \cdot 10\ byte + vis + \frac{ipas}{2} + \lceil \log_2 N \rceil)$
- calculations: an address offset calculation for the prefix table access, one address offset calculation for accessing the following subtrie or the routing information per subtrie, two comparisons per node. In the worst-case, three subtries must be passed and the length of the path is 15 nodes. That is, up to 34 operations may be needed per packet.
- delay per packet: $5 \cdot lat_{ipr} + 34 \cdot clk^{-1}$ with $g = max(addr_{ipr}, 10\ Byte, vis + \frac{ipas}{2} + \lceil \log_2 N \rceil)$. Again, accesses may be divided into small address field accesses of the precomputed table ($g = addr_{ipr}$) and longer accesses of the subtries and the routing information ($g = max(10\ Byte, vis + \frac{ipas}{2} + \lceil \log_2 N \rceil)$). The formula can then be refined to $lat_{ipr,addr} + 4 \cdot lat_{ipr} + 34 \cdot clk^{-1}$.

11.2 Example

See Tab. 11.

12 Design space exploration of the memory subsystem

The results of the preceding sections can now be used to explore the number of memory chips and buses required. Memory controllers for SRAM and SDRAM buses are grouped around a single CPU core. Two different system settings are investigated, one for the 155 Mbit line rate and another for a 622 Mbit line rate. The settings are given in Tab. 12.

The data dependencies of the building blocks are visualized in Fig. 5. There are four potential critical paths. In order to explore the design space, every building block is mapped onto one of four possible

Fixed model parameters		
$ipas$	64 bit	IP source plus destination address field size
$ipps_{min}$	40 byte	minimal packet size
vis	28 bit	VPI/VCI field size
con	14 bit	flow context information
$addr_{ipr}$	22 bit	router RAM addresses
clk	200 MHz	CPU clock

Model parameters					*Patricia trie, average case*				*compr. prefix trie*			
r	nr	N	lat_{addr}	lat_{con}	sp	th	ops	del	sp	th	ops	del
[Mbit/s]			[ns]	[ns]	[Mbit]	[Mbit/s]	[10^6]	[ns]	[Mbit]	[Mbit/s]	[10^6]	[ns]
155	1000	256	50	70	0.126	167.2	4.8	620	3.0	152.4	15.7	500
155	1000	256	20	30	0.126	167.2	4.8	280	3.0	152.4	15.7	310
155	1000	512	50	70	0.127	167.7	4.8	620	3.0	152.9	15.7	500
155	1000	512	20	30	0.127	167.7	4.8	280	3.0	152.9	15.7	310
155	40000	256	50	70	5.04	244.8	7.8	950	14.5	152.4	15.7	500
155	40000	256	20	30	5.04	244.8	7.8	430	14.5	152.4	15.7	310
155	40000	512	50	70	5.07	245.3	7.8	950	14.5	152.9	15.7	500
155	40000	512	20	30	5.07	245.3	7.8	430	14.5	152.9	15.7	310
622	1000	256	50	70	0.126	671.0	19.4	620	3.0	611.7	63.0	500
622	1000	256	20	30	0.126	671.0	19.4	280	3.0	611.7	63.0	310
622	1000	512	50	70	0.127	672.9	19.4	620	3.0	613.6	63.0	500
622	1000	512	20	30	0.127	672.9	19.4	280	3.0	613.6	63.0	310
622	40000	256	50	70	5.04	982.5	31.1	950	14.5	611.7	63.0	500
622	40000	256	20	30	5.04	982.5	31.1	430	14.5	611.7	63.0	310
622	40000	512	50	70	5.07	984.3	31.1	950	14.5	613.6	63.0	500
622	40000	512	20	30	5.07	984.3	31.1	430	14.5	613.6	63.0	310

Table 11: IP router example.

RAM buses: two buses for DRAMs and two buses for SRAMs. Then, the required capacity and thus the number of memory chips can be determined per RAM bus. Every bus needs its own memory controller. Finally, the costs and latency for such a configuration can be derived. All possible 4^8 configurations have been calculated and redundant as well as incorrect solutions due to latency constraints of critical paths and the RAM buses have been discarded. The cell clock cycle on the ATM line has been used as delay bound.

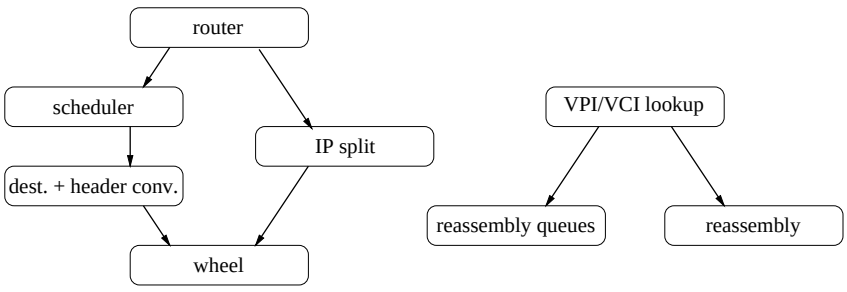

Figure 5: Data flow graph of the access node.

Fixed model parameters

common parameters		
cs	424 bit	cell size
hs	40 bit	cell header size
$ipps_{max}$	1536 byte	maximal packet size
$ipps_{min}$	40 byte	minimal packet size
vis	28 bit	VPI/VCI field size
con	14 bit	context information
$addr_{(.)}$	22 bit	RAM addresses
clk	200 MHz	CPU clock frequency
N	512	number of nodes along line
nc	2^{13}	max. number of connections

main memory system		
ov_{SDRAM}	40 ns	access overhead, SDRAM @ 100 MHz
ov_{SRAM}	12 ns	access overhead, SRAM @ 166 MHz
$cost_{SDRAM}$	10 \$	price of a SDRAM chip with 128 Mbit capacity
$cost_{SRAM}$	8 \$	price of a SRAM chip with 4 Mbit capacity
$cost_{mc,SDRAM}$	20 \$	price of a memory controller for SDRAMs
$cost_{mc,SRAM}$	$\frac{1}{3} \cdot cost_{mc,SDRAM}$	price of a memory controller for SRAMs

Wheel framing stage, lookup implementation		
fs	2048	frame size

MD-SCFQ + WFQ approx.		
r_{min}	128 Kbit	minimally reservable rate
r_{max}	10^8 Mbit	maximally reservable rate
ts	32 bit	scheduling tag size
cp	16 bit	backlog counter precision
$nsic$	16	number of service interval classes

FP scheduler, vector lookup implementation		
pcs	64	number of priority classes

VPI/VCI cell removal		
$ne + n_{ubr}$	2^{13}	w-c number of UBR and ending connections

reassembly		
ne	2^{13}	w-c number of connections ending at current node

router		
nr	40000	number of routing table entries
$ipas$	64 bit	IP source + destination address field size

Table 12: Design space exploration parameters.

155 Mbit line rate: MD-SCFQ combined with the data structure of the WFQ approximation as well as fixed priority are considered for the scheduler block. The original WFQ variant however cannot be applied since even by using SRAMs WFQ may miss the delay bound given by a cell clock cycle. The VPI/VCI lookup is implemented by a Patricia tree. The delay bound is 2730 ns. In Fig. 6 the solutions

are plotted through the design space defined by costs and latency. The Pareto optimal solutions are emphasized as well as the region they dominate. For the configurations using a fixed priority scheduler the following six Pareto points from 16000 correct solutions have been determined, see Tab. 13.

Table 13: Pareto solutions using a FP scheduler.

Objectives		*#chips*		*#buses*	
costs [$]	*latency* [ns]	*SDRAM*	*SRAM*	*SDRAM*	*SRAM*
84	1165	3	1	2	1
72	1450	3	2	1	1
64	1590	3	1	1	1
442	865	1	50	1	2
102	975	3	5	1	2
466	765	1	53	1	2

The configurations using more than five SRAM chips are not feasible for a PCB layout. The remaining configurations cope well with the delay bound and therefore have resources left for further building blocks. The calculation requirement of 39 Mops is moderate. However, one has to keep in mind the bad delay characteristics of fixed priority scheduling.

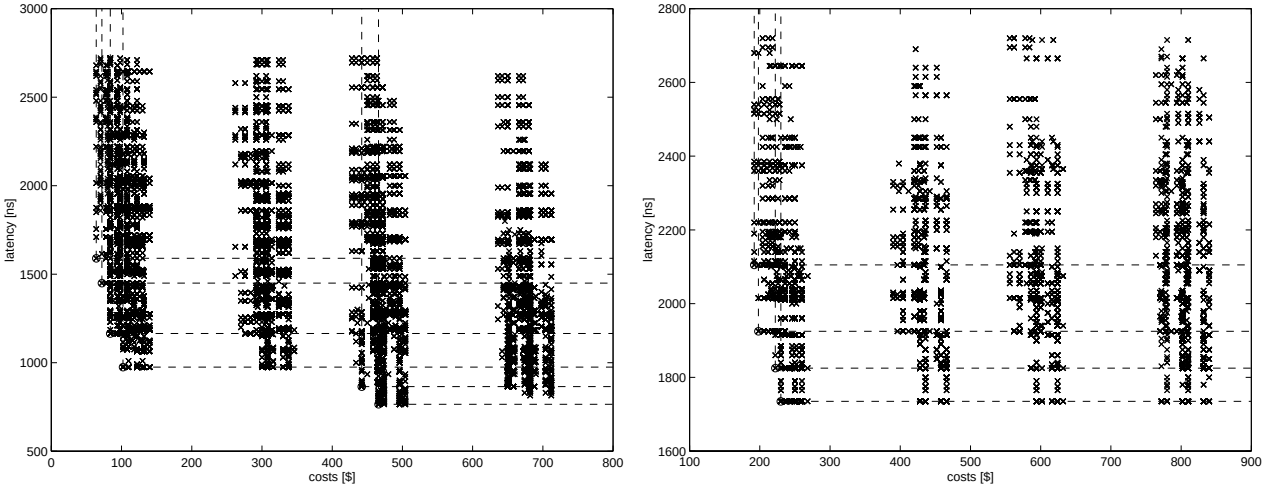

Figure 6: Memory subsystem design space exploration of the access node.

The Pareto solutions for the access node using a WFQ approximation data structure with 128 subdivisions of service intervals together with MD-SCFQ scheduling are shown in Tab. 14 (from about 8000 correct solutions):

62 Mops are required for calculations. The high amount of SRAMs needed is caused by the throughput demand of the WFQ approximation data structure. All correct solutions need at least 16 SRAM chips. The requirements of the data structure can be reduced by decreasing the amount of subdivisions from 132 to 32 accepting a higher delay penalty of the scheduling algorithm. Then, the following feasible Pareto solutions are found, see Tab. 15.

Table 14: Pareto solutions using MD-SCFQ and WFQ approx. data structure with 128 subdivisions.

Objectives		*#chips*			*#buses*		
costs [$]	*latency* [ns]	*SDRAM*	/	*SRAM*	*SDRAM*	/	*SRAM*
192	2105	3	/	17	1	/	1
198	1925	3	/	17	1	/	2
222	1825	3	/	20	1	/	2
230	1735	3	/	21	1	/	2

Table 15: Pareto solutions using MD-SCFQ and WFQ approx. data structure with 32 subdivisions.

Objectives		*#chips*			*#buses*		
costs [$]	*latency* [ns]	*SDRAM*	/	*SRAM*	*SDRAM*	/	*SRAM*
96	2055	3	/	5	1	/	1
102	1875	3	/	5	1	/	2
126	1775	3	/	8	1	/	2
134	1685	3	/	9	1	/	2

Note that all Pareto-optimal configurations need both types of memory. In the MD-SCFQ case, even all correct solutions require SDRAMs and SRAMs. A stand-alone SDRAM based main memory system cannot cope with the demands of the complex scheduling stages. The number of memory chips can be reduced by the transition to higher capacity RAMs. SDRAMs chips are currently available up to a capacity of 256 Mbit and SRAMs up to 8 Mbit respectively.

622 Mbit line rate: For this point of operation the choice of RAM components reduces dramatically because of the tighter delay bound of 680 ns. The more complex scheduling building blocks cannot be used due to their latency overhead. Thus, only configurations using a fixed priority scheduler have been investigated. Moreover, the VPI/VCI lookup stage must be implemented with a memory space consuming 2^8 trie in order to cope with the delay bound. However, only two correct solutions and a single Pareto solution have been determined, see Tab. 15.

Table 16: Pareto solution using a FP scheduler.

Objectives		*#chips*			*#buses*		
costs [$]	*latency* [ns]	*SDRAM*	/	*SRAM*	*SDRAM*	/	*SRAM*
554	675	1	/	64	1	/	2

Even though this solution only requires 119 Mops for computations, it is virtually not feasible because of the high chip count. Thus, for the 622 Mbit rate, one may think of pipelining the building blocks of the access node. Distinct memory chips are then required for every building block and the calculation requirements increase accordingly.

13 Conclusion

In this report we have modeled the building blocks of an IP over ATM shared medium access node with enhanced functionality for routing and QoS distinction of IP flows. In order to evaluate the impact of the different tasks such as routing and fair scheduling on the implementation of the access node on a single processor module, state of the art algorithms have been analyzed in terms of memory and computing requirements under realistic workloads. The requirements have then been mapped onto main memory and CPU models in order to determine worst-case latency and memory usage bounds. These bounds have been used to derive feasible configurations of the processor module. The results for our case study can be summarized as follows:

- Supporting a 155 Mbit ATM line rate, up to 2^{13} concurrent connections, a fair queuing scheduler, a backbone routing stage with up to 40000 routing entries, and the buffering of at least 2^{15} packets only requires a moderately clocked CPU core (200 MHz) together with two memory buses and four to eight memory chips.
- The fair queuing stage is the performance bottleneck of the shared processor module implementation. The data structures of this stage must therefore be mapped onto SRAMs in order to cope with the delay bounds given by the worst-case analysis of suitable algorithms. The queuing algorithms cannot be implemented stand-alone above a supported rate of about 250 Mbit assuming worst-case 40 Byte packets.
- All Pareto-optimal configurations of the main memory subsystem (even by replacing the fair scheduler by a fixed priority scheduler) require SRAMs and SDRAMs and therefore two buses and memory controllers.
- The costs of the memory subsystem (memory chips and controllers) for the whole access node supporting an ATM line rate of 155 Mbit can be kept below 100 $.
- At a 155 Mbit ATM line rate, the CPU and memory resources show a worst-case utilization of 75 % leaving room for additional building blocks or slightly higher line rates.
- The described routing stage can perform up to $2 \cdot 10^6$ lookups/s stand-alone using cheap SDRAM technology independent on the distribution of addresses and accesses on the backbone routing table.

The models derived in this report can be used for high level design space explorations in order to find a suitable trade-off between the complexity of algorithms, the accuracy of guaranteed delay bounds, the hardware costs of the system, and feasible hardware configurations. The described access node may be implemented up to a rate of 200 Mbit on a single processor module. The system can be further accelerated by considering on-chip RAM, pipelining of the distinct building blocks, and additional special hardware, e.g. for fast searching and sorting in priority queues.

References

[1] Jon C. R. Bennett and Hui Zhang. Hierarchical packet fair queueing algorithms. *IEEE/ACM Transactions on Networking*, 5(5):675–689, October 1997.

[2] Jon C.R. Bennett and Hui Zhang. WF^2Q: worst-case fair weighted fair queueing. In *IEEE INFOCOM '96, The Conference on Computer Communications*, volume 1, pages 120–128, 1996.

[3] H. Jonathan Chao, Yau-Ren Jenq, Xiaolei Guo, and Cheuk H. Lam. Design of packet-fair queuing schedulers using a RAM-based searching engine. *IEEE Journal on Selected Areas in Communications*, 17(6):1105–1126, June 1999.

[4] Fabio M. Chiussi and Andrea Francini. Minimum-delay self-clocked fair queueing algorithm for packet-switched networks. In *Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications*, volume 3, pages 1112–1121. IEEE, 1998.

[5] Ki-Ho Cho and Hyunsoo Yoon. Design and analysis of a fair scheduling algorithm for QoS guarantees in high-speed packet-switched networks. In *ICC '98 IEEE International Conference on Communications*, volume 3, pages 1520–1525, 1998.

[6] Yen-Ping Chu and E-Hong Hwang. A new packet scheduling algorithm: minimum starting-tag fair queueing. *IEICE Transactions on Communications*, E80-B(10):1529–1536, October 1997.

[7] Jorge A. Cobb, Mohamed G. Gouda, and Amal El-Nahas. Time-shift scheduling-fair scheduling of flows in high-speed networks. *IEEE/ACM Transactions on Networking*, 6(3):274–285, June 1998.

[8] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. In *Computer Communication Review, ACM SIGCOMM*, volume 27, pages 3–14, October 1997.

[9] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. *Internetworking: Research and Experience*, 1(1):3–26, September 1990.

[10] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[11] S. Jamaloddin Golestani. A self-clocked fair queueing scheme for broadband applications. In *INFOCOM 94*, volume 2, pages 636–646. IEEE, June 1994.

[12] Pawan Goyal, Harrick M. Vin, and Haichen Cheng. Start-time fair queuing: a scheduling algorithm for integrated services packet switching networks. *Computer Communication Review*, 26(4):157–168, October 1996.

[13] Matthias Gries and Andreas Romer. Performance Evaluation of Recent DRAM Architectures for Embedded Systems. Technical Report 82, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, November 1999.

[14] Pankaj Gupta, Steven Lin, and Nick McKeown. Routing lookups in hardware at memory access speeds. In *INFOCOM'98*, volume 3, pages 1240–1247. IEEE Computer and Communications Societies, 1998.

[15] Nen-Fu Huang and Shi-Ming Zhao. A novel IP-routing lookup scheme and hardware architecture for multi-gigabit switching routers. *IEEE Journal on Selected Areas in Communications*, 17(6):1093–1104, June 1999.

[16] IETF Internet Engineering Task Force. Differentiated and integrated services working groups. *http://www.ietf.org*.

[17] Douglas W. Jones. An empirical comparison of priority-queue and event-set implementations. *Communications of the ACM*, 29(4):300–311, April 1986.

[18] Donald E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, 2 edition, 1998.

[19] Butler Lampson, Venkatachary Srinivasan, and George Varghese. IP lookups using multiway and multicolumn search. In *INFOCOM'98*, volume 3, pages 1248–1256. IEEE Computer and Communications Societies, 1998.

[20] Sung-Whan Moon, Kang G. Shin, and Jennifer Rexford. Scalable hardware priority queue architectures for high-speed packet switches. In *Third IEEE Real-Time Technology and Applications Symposium*, pages 203–212, 1997.

[21] Donald R. Morrison. PATRICIA - practical algorithm to retrieve information coded in alphanumeric. *Journal of the Association for Computing Machinery*, 15(4):514–534, October 1968.

[22] Stefan Nilsson and Gunnar Karlsson. IP-address lookup using LC-tries. *IEEE Journal on Selected Areas in Communications*, 17(6):1083–1092, June 1999.

[23] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[24] Jennifer L. Rexford, Albert G. Greenberg, and Flavio G. Bonomi. Hardware-efficient fair queueing architectures for high-speed networks. In *IEEE INFOCOM '96, Fifteenth Annual Joint Conference of the IEEE Computer Societies*, volume 2, pages 638–646. IEEE Comput. Soc. Press, 1996.

[25] Robert Rönngren and Rassul Ayani. A comparative study of parallel and sequential priority queue algorithms. *ACM Transactions on Modeling and Computer Simulation*, 7(2):157–209, April 1997.

[26] Hanrijanto Sariowan, Rene L. Cruz, and George C. Polyzos. SCED: A generalized scheduling policy for guaranteeing Quality-of-Service. *IEEE/ACM Transactions on Networking*, 7(5):669–684, October 1999.

[27] M. Shreedhar and George Varghese. Efficient fair queuing using Deficit Round-Robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, June 1996.

[28] Keith Sklower. A tree-based packet routing table for Berkeley UNIX. In *USENIX Winter Conference*, pages 93–103, January 1991.

[29] Donpaul C. Stephens, Jon C.R. Bennett, and Hui Zhang. Implementing scheduling algorithms in high-speed networks. *IEEE Journal on Selected Areas in Communications*, 17(6):1145–1158, June 1999.

[30] Dimitrios Stiliadis and Anujan Varma. Efficient fair queueing algorithms for packet-switched networks. *IEEE/ACM Transactions on Networking*, 6(2):175–185, April 1998.

[31] Dimitrios Stiliadis and Anujan Varma. Rate-proportional servers: a design methodology for fair queueing algorithms. *IEEE/ACM Transactions on Networking*, 6(2):164–174, April 1998.

[32] Ion Stoica, Hui Zhang, and T. S. Ng. A hierarchical fair service curve algorithm for link-sharing, real-time and priority services. *Computer-Communication-Review*, 27(4):249–262, October 1997.

[33] Subhash Suri, George Varghese, and Girish Chandranmenon. Leap forward virtual clock: a new fair queuing scheme with guaranteed delays and throughput fairness. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, 1997.

[34] Henry Hong-Yi Tzeng and Tony Przygienda. On fast address-lookup algorithms. *IEEE Journal on Selected Areas in Communications*, 17(6):1067–1082, June 1999.

[35] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable high speed IP routing lookups. *Computer Communication Review, ACM SIGCOMM*, 27(4):25–36, October 1997.

[36] Hui Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–1396, October 1995.

[37] Hui Zhang and Domenico Ferrari. Rate-controlled service disciplines. *Journal of High Speed Networks*, 3(4):389–412, 1994.