Joint Evaluation of Architecture and Behavior for Network Processing Systems

Matthias Gries, *Member, IEEE*

Abstract—During early development phases of a new design, decisions have to be made that have highest impact on the final quality of the system, including the selection of algorithms to solve a given problem and the binding of functionality to processing resources. Yet, common engineering practice is to rely on back-of-the-envelope calculations and design experience in order to derive a decision at this point. We use a case study in the domain of access networks to reveal the benefit of quantitative evaluation in this early concept phase of the design. In this domain, where the growing deployment of fast subscriber lines accelerates the need for Quality of Service (QoS) distinction of voice, video, and data services, a feasible and economic implementation of QoS is mandatory. By simulating algorithm behavior and hardware performance together, we show that we are more sensitive to queue management than to packet scheduling with respect to processing requirements. Moreover, the complexity for sorting and searching, as well as buffer management is not a concern, which is against our initial intuition as designers. We also point out quantitatively that the QoS behavior (fairness, packet latency) particularly depends on the selection of the link scheduler. We show that a Round-Robin based scheduler as used by access nodes today puts QoS in jeopardy, even though only a relatively small number of traffic flows is distinguished. A much better choice is the use of a simple Weighted Fair Queuing approximation, while the difference in implemention complexity is negligible in our access network setup.

Index Terms—network processing, policing, queue management, scheduling, design space exploration, system level design, triple play, customer premises equipment, quality of service.

1 Introduction

THE growing installation of fast network subscriber lines to homes and businesses enables new service models and usage scenarios. Recently, the ”triple play” setup, i.e. the convergence of voice, video, and data services over one network cable offered by a single service provider, has become the driving force behind Quality of Service (QoS) deployment in the access network. With access network we broadly mean the network infrastructure between the core network, where data forwarding takes place, and the Customer Premises Equipment (CPE), where network traffic is generated and consumed. A simplified setup is shown in Figure 1. More detailed background material can be found in [1], [2]. CPE comprises all end devices engineered for the customer's home under her/his control, such as residential gateways, modems, and VoIP phones. Subscriber lines from different customers are aggregated by Digital Subscriber Line Access Multiplexers (DSLAMs). DSLAMs can prioritize traffic with respect to different customers and network flows. The expected QoS is subject to Service Level Agreements (SLAs). In the triple play case, voice traffic is given the highest priority due to the ear's sensitivity to delay and jitter. The required network volume for voice traffic is relatively low. Video traffic for television broadcasts or Video on Demand schemes is given medium priority, assuring low loss rates and low latency. Finally, Internet traffic

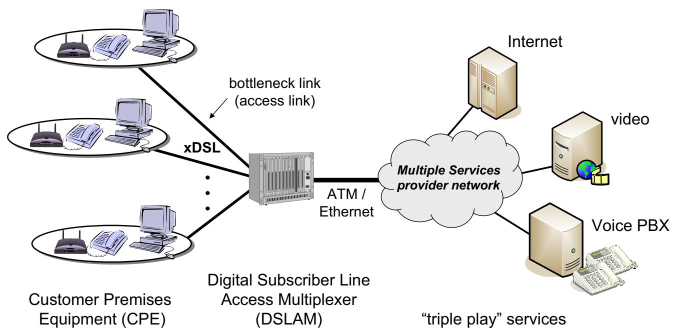

bottleneck link (access link)
xDSL
Internet
video
Multiple Services provider network
ATM / Ethernet
Voice PBX
Customer Premises Equipment (CPE)
Digital Subscriber Line Access Multiplexer (DSLAM)
“triple play” services

Fig. 1. Triple Play scenario of converged voice, video, and data services.

gets whatever remains on the customer's link to the service provider (best effort service). In this way, greedy Internet traffic cannot jeopardize the delivery of video content.

On the one hand, CPE devices are customized by service providers in order to bind customers to their preferred services. A flexible platform architecture enables reuse by several service providers. On the other hand, CPE devices constitute a financial overhead for the service providers that must be minimized. Last but not least, CPE must show power efficiency since these device are often used always-on. A customer will not accept a 400W power supply and noisy cooling in her/his living room if, in principle, the functionality can also be provided by her/his personal computer. Consequently, CPE must be designed under tight constraints on power dissipation and costs, while other design criteria, such as certain flexibility/programmability and performance, must be maintained. The same statements also hold for DSLAMs, although at a larger scale. Implementing

• *Present address: Infineon Technologies, Access Communication Solutions, 81726 Munich, Germany.*
The work was mainly performed at Swiss Federal Institute of Technology (ETH) Zürich, Computer Engineering and Networks Laboratory (TIK) 8092 Zürich, Switzerland.
email: gries@computer.org

Unpublished technical report, edited June 4th, 2006. This report summarizes and updates the evaluation in [58].

QoS means a certain overhead per port (customer) that must be minimized. Still, DSLAMs aggregating several thousand customers must fit into industry size 19" racks, which also has implications on the tolerable power dissipation.

This is why the selection of the "right" algorithms for implementing QoS plays an important role in finding a reasonable and economically viable solution in the design space of access network systems. In this context, the main contributions of this paper are:

- We give an overview of the packet processing tasks required for providing Quality of Service. Although the behavior of individual tasks is well understood, we reveal by simulation that the interplay of algorithms for different tasks must match in order to implement QoS with anticipated behavior.
- We show that the implementation of sophisticated algorithms can make sense in CPE devices, although the number of traffic flows is small compared with other access and backbone network devices. The algorithms used at the CPE side are usually considerably less complex than the ones used in the remaining network.
- We quantify design trade-offs with respect to the complexity of implementing the algorithms versus their QoS properties. We show that, by employing reasonable approximations of state-of-the-art algorithms, the QoS properties in access devices can considerably be improved while maintaining the complexity of currently deployed algorithms.

The evaluations are conducted by simulation of functional-correct algorithm models together with performance models of hardware building blocks. The deployment of these algorithms makes sense at both sides of the subscriber line, which constitutes a bottleneck resource where access must be arbitrated. That means, in downstream direction (from the service provider to the customer) particularly the DSLAM is in charge with assuring QoS. In upstream direction (from the customer to the network), the CPE has to guarantee QoS distinction of traffic flows.

Although not the main focus of this paper, We also give a short overview of our evaluation method for completeness. Our approach modifies current state-of-the-art system-level design methods so that time-dependent behavior can be modeled, as required by some of the scheduling algorithms considered in this paper.

The main purpose of this paper is to expose corner cases of the design space for access devices, looking at algorithm behavior and complexity for deploying Quality of Service. We emphasize how these system level design decisions can affect the quality and feasibility of flexible platform architectures and thus point out the importance of judicious benchmarking from the beginning of the design process.

The paper is organized as follows. In the next section we give an overview of packet processing functionality needed for implementing QoS. Section 3 continues with a description of our modeling and evaluation methodology based on process networks, discrete event simulation, and performance models. In Section 4, our case study of implementing QoS at a bottleneck link at an access network node reveals design trade-offs with respect to implementation complexity and QoS behavior. Section 5 concludes the paper.

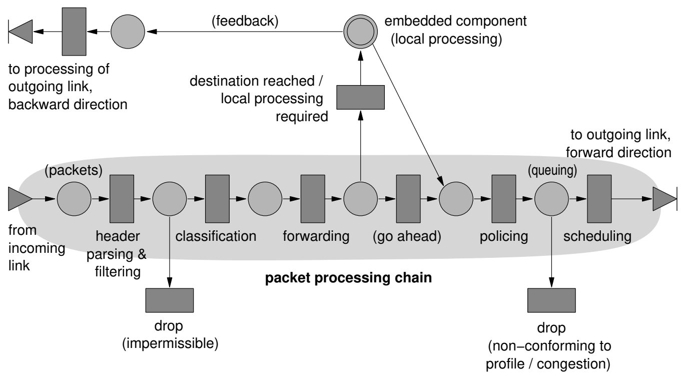

(feedback)
embedded component (local processing)
to processing of outgoing link, backward direction
destination reached / local processing required
to outgoing link, forward direction
(packets)
(queuing)
from incoming link
header parsing & filtering
classification
forwarding
(go ahead)
policing
scheduling
packet processing chain
drop (impermissible)
drop (non–conforming to profile / congestion)

Fig. 2. The packet processing chain for the QoS forwarding path (Petri net model).

2 Quality of Service Algorithms

We give an overview of the essential functionality for providing Quality of Service (QoS) in this chapter. A review of the algorithms applied in related work is used for the selection of a representative set of implementations for the case study described later in this paper. We mainly focus on the packet processing functionality in the data plane. Control plane tasks, such as signaling, are usually not bounded by communication and processing resources. An overview of charging and accounting, as well as signaling for access networks can be found in [1], [2] and the references therein.

The IP network layer is the lowest layer of the OSI reference model that considers end-to-end transmission of data. For each incoming IP packet a network node must decide to which node the packet will be forwarded next. The decision is based on the information stored in the IP packet header and additional state information in the node itself. Packet processing includes parsing the packet header, classification of the packet so as to assign a packet to a QoS class, determination of the next hop (forwarding), check of Service Level Agreements (policing), queuing, and finally link scheduling, as visualized in the flow in Fig. 2 (Petri net representation).

The different packet processing tasks are introduced in the following subsections. We distinguish between tasks that are required to update header fields and identify traffic flows and tasks that directly affect the QoS experienced by a flow. The former set of tasks is briefly described to give a comprehensive overview of the required functionality. Tasks in this set do not actively change the QoS behavior of a flow, but provide the necessary processing to enable QoS distinction. The evaluation of QoS algorithms later in this paper will be based on the latter set of tasks, i.e. policing, queue management, and link scheduling.

2.1 Overview of Required Packet Processing Tasks

By parsing the header of an incoming packet, information, such as the length of the packet, the destination address, and the protocol type, is extracted. A filter stage with a small set of rules then decides based on this header input whether the packet is allowed to pass further processing stages or whether it should be dropped immediately. In case of admission, a more elaborate classification stage uses the extracted header fields to associate the packet with its context information, such as the corresponding Quality of Service (QoS) class – a flow identifier – and the reserved rate. A forwarding stage, which may be combined with the classifier, uses the destination address of the packet to determine whether the packet should be passed on to a particular outgoing link or to further internal processing tasks. Further internal processing may be required if the destination is reached or some higher protocol layers must be processed. If it is decided to forward the packet to an outgoing link, the packet will be handed to the policer. The policer uses the context information assigned by the classifier to check whether a packet complies to a defined traffic profile of the corresponding flow. A traffic profile may specify properties like the maximum burstiness and the rate of incoming traffic. A profile is subject of a Service Level Agreement (SLA) between a customer and a service provider. An SLA states that as long as traffic complies to a defined profile, the provider will ensure a certain level of service, for instance, in terms of delay and loss. Thus, the profiler marks a packet as conforming or as non-conforming to a flow's profile. Non-conforming packets may immediately be dropped. Before the packet can finally be transmitted through the outgoing link, it must be queued until the link scheduler chooses the packet for transmission. The scheduling policy depends on the header and context information assigned to the packet (e.g., packet length, SLA, assigned QoS class).

2.2 Header Processing

Tasks in this subsection provide the necessary functionality to enable QoS, but do not actively change the QoS properties of a flow. After having parsed the packet header, field information is available by which the incoming packet can be characterized. A packet may pass filtering, classification, and forwarding stages to be routed to the next hop.

2.2.1 *Filtering*

Since every incoming packet is examined in this stage, only a small number of rules are applied to match line speed. The rules assure that only authorized packets pass through the following processing elements and that packets, which are directed to the current node, are taken out of the traffic stream. The latter task can also be performed by the forwarding stage. Rules can span several header fields (dimensions).

2.2.2 *Classification*

This stage resembles the filtering stage, although a higher number of rules are usually applied. Context information is assigned to a packet depending on the header fields and according to a set of possibly overlapping rules. Rules can be weighted by, e.g., priority or cost. Accounting, billing, and QoS-aware packet handling are enabled in this way. The classification problem can be solved by search approaches that are implemented by bitmap intersection [3], walking through so-called fat inverted segment trees [4], or heap on trie data structures [5]. These algorithms provide different – sometimes configurable – trade-offs between search and update times, as well as the storage requirements. Since they employ general data structures, their worst-case behavior does not depend on the actual range distribution. Opposed to that, the method of hierarchical cuttings [6] adaptively subdivides the range space into subproblems which can be solved by linear search. Finally, tuple space search [7] starts with a linear search in a rule set, whose number of rules has been reduced by heuristics, and then jumps into a hash table. Feldmann and Muthukrishnan present a framework in [4] by which the general multi-dimensional packet classification problem can be mapped to multiple instances of a longest prefix match problem, thus reusing forwarding solutions.

2.2.3 *Forwarding*

Forwarding provides the actual routing decision. Using the destination address, the outgoing link to the next hop is determined. IPv4 forwarding applies the Classless Inter Domain Routing (CIDR [8]) address scheme. Forwarding algorithms trade off fast search times for small memory footprint. A Patricia trie [9] is a general data structure that has been used for forwarding tasks in a number of software implementations, e.g. [10], and is a reasonable option for small routing tables. One extreme is to precompute the largest reasonable lookup table in order to find a match with a small number of memory accesses, see [11]. The other extreme is to compress the data structure as much as possible at the expense of a higher number of memory accesses [12]. Balanced solutions are level compressed tries [13], multi-way search [14], and subtrie compression [15].

2.3 Policing

After classification the context information of a packet is available, such as the traffic flow. Service guarantees can now be checked by verifying Service Level Agreements (SLAs) between customers and the service provider. This is done by measuring the flow's actual traffic profile. If the current packet is within the guaranteed profile, the packet will be processed without any restriction and marked as conforming to the SLA. If the profile is not kept, the packet can be dropped or marked as non-conforming and forwarded with degraded service. In addition, a packet may be delayed before policing in order to shape the flow according to a profile. This approach will be advantageous if there is some background knowledge that the flow entered the network according to that profile and has been reshaped by the characteristics of intermediate network nodes. If the shaper and the policer use the same profile for the flow, the shaper may take over the task of marking.

The TSpec [16] has been introduced as traffic specification by the Internet community to treat QoS reservations. A TSpec is defined by peak rate p, average rate r, burstiness b, maximum packet size M, and minimal policed unit of the traffic flow.

2.3.1 *Metering*

A token bucket can be used as traffic meter, defined by its token capacity and the token fill rate. Traffic may be marked as conforming to a TSpec by checking two token buckets running in parallel. One bucket has the size M and is filled by the peak rate p, the other one has the capacity b and is filled by the average rate r. A packet will be allowed to pass the dual token bucket if there is a sufficient amount of tokens in each of the buckets. Then, the corresponding amount of tokens is taken from both buckets, e.g. measured in Bytes. In order to enable a graceful service degradation if traffic does not meet a certain profile but is within a less restrictive profile, nested token buckets can be used. In its simplest form [17] by nesting two token buckets with profiles (b, r) and (B, R), $b \leq B$ and $r \leq R$, a packet will be marked as conforming to the (b, r) profile if there are enough tokens in both buckets. Then, tokens are taken from both buckets and the packet is marked for premium service. If, however, the packet does not fit into the (b, r) but only into the (B, R) profile, the packet is marked for degraded service and tokens are only taken from the (B, R) bucket. At last, if there are not enough tokens in the (B, R) bucket, the packet will be marked as non-conforming and tokens will not be taken from any bucket.

2.3.2 Shaping

All the mechanisms described for metering can be used as a basis for a corresponding shaper entity. The only difference is that a packet is never dropped – unless the shaper runs out of buffer space – but is delayed until it finally conforms to a given profile.

For the case study described later, we use token buckets for individual traffic flows and nested token buckets for aggregated flow classes that contain several flows.

2.4 Queue Management

After a packet has been admitted to transmission, it must be buffered in the system until it will either be chosen by the link scheduler for transmission or be discarded in case of a congested link. It is the responsibility of a queue manager to operate the packet storage space, which may include dynamically allocating and deallocating memory to store or release packets (buffer management), as well as coping with congestion, i.e., choosing packets to discard.

Ideally, the behavior of the different flows should be isolated from each other. Packets marked as conforming to their profiles should always be stored regardless of other greedy traffic sources. Surplus storage space should be shared in a fair manner. In order to balance the separation of flows and the number of flows that can be managed, different approaches are possible:

- *Single queue:* Packets from all flows are enqueued in the order they arrive at the queue manager without storing any additional per flow state. Although a high number of flows can be aggregated this way since one does not depend on any per flow state information, the search for a packet of a particular flow needs an exhaustive linear search among all enqueued packets.
- *Separate queues:* There is a separate FIFO queue for each flow. The memory space is statically subdivided into parts that are exclusively assigned to distinct flows. The expense for organizing queues can be kept at a moderate level, e.g. by employing ring queue data structures based on arrays. While this approach achieves perfect isolation of flows, storage resources may be wasted since flows are not allowed to exploit memory that is currently unused by other flows.
- *Shared memory:* FIFO queues for the flows are organized as linked lists of data segments of a defined size. The contents of a packet may be distributed over several segments. Memory space is dynamically allocated and released for every packet by maintaining a list of free segments. By defining individual thresholds of space utilization for each flow according to some reservation rule, a flow can be protected against other flows as long as its occupation of the memory is below its threshold. Memory beyond these thresholds can be shared arbitrarily at the expense of the preservation of more complex data structures and per-flow state information.

Independently of the organization of the memory, the congestion behavior of the queue manager can be determined using the following guidelines:

- *Congestion avoidance:* A congestion may be avoided by preventively discarding packets depending on the system's state before the appearance of congestion.
- *Congestion recovery:* Packets are dropped in times of congestion to avoid deadlocks.

In order to succeed, the queue manager may apply

- *Rejection of packets at arrival:* Packets may not be allowed to enter the queue manager depending on the state of the system.
- *Pushing out already stored packets:* Packets that have already been enqueued in the queue manager are discarded at the arrival of new packets. Common actions are dropping packets from the front or from the tail of a queue, as well as random selection of a packet [18].

2.4.1 Congestion avoidance

Common implementations of congestion avoidance include Early Packet Discard (EPD [19]) and Random Early Detection (RED [20]). Incoming packets will be immediately dropped by EPD if the current size of the queue passes a fixed threshold. RED starts to drop incoming packets with a defined probability as soon as the average size of allocated memory exceeds a given threshold. The probability to be dropped increases linearly with the average memory size. If a second threshold is passed, every incoming packet will be dropped. Flow random early drop (FRED [21]) employs per flow state information – in particular queue lengths for individual flows – to more fairly drop packets from flows depending on individual memory utilization.

2.4.2 Congestion recovery

Congestion recovery mechanisms – partly extended by using policing information and congestion avoidance ideas – have been implemented in Longest Queue Drop (LQD [22]). LQD simply pushes out packets from the currently longest flow queue. Assuming an equal reservation of resources for all flows, flows are protected against misbehaving flows that show a longer backlog of packets and thus experience higher loss rates. However, one must keep track of the longest queue in the system. Extended Threshold Policy (ETP [23]) pushes out packets in order to cope with congestion. Congestion may be avoided because all packets that have been marked as non-conforming by the policer and that are currently stored above a defined threshold will be discarded if a packet marked as conforming arrives. Note that a packet arrival may initiate several packet discards. Extended Simulated Protective Policy (ESPP [23]) also pushes packets out to recover from congestion. A second queuing system is maintained as a reference which only deals with conforming traffic. The goal is to ensure that the main system offers the same service for conforming traffic as the reference system at any point of time.

For the case study described later we choose three representative queue managers (QMs). The most complex one relies on congestion recovery and maintains separate queues for each flow. Also, the handling of conforming traffic is split from handling surplus traffic (marked as non-conforming) while maintaining in-order packet transmissions. Conforming traffic is therefore protected from other traffic and we call this version a fair QM. The second QM still uses separate queues for conforming traffic, whereas non-conforming traffic from all flows is aggregated into a single queue to reduce per-flow state maintenance. This means, conforming traffic (green traffic) is still well protected, whereas non-conforming (yellow) traffic competes FCFS for buffer space. We call this version a central yellow queue QM. The third QM is based on congestion avoidance by applying RED [20] to the central yellow queue of the preceding variant. The third variant is taken into account to compare the implementation complexity of RED versus fair QM.

2.5 Link Scheduling

A link scheduler is an arbiter that decides which of the buffered packets will be transferred next through a bottleneck link of a network node. The scheduler uses further data, such as the system state, SLAs, or traffic characteristics metered by a policer, to guide and support its decision. Schedulers are considered to be fair if surplus bandwidth is distributed to backlogged flows in proportion to their reservation, i.e., they should not give preference to any flow.

Dynamic priority-driven schedulers provide the necessary accuracy to cope with fairness and separation/protection of flows under memory and delay constraints. Since they use priorities generated at run-time, packets must be dynamically sorted accordingly. The sorted data structure is often called a *priority queue*. That means, not only the calculation of priority values is needed, but also a sorted data structure must be maintained. It is sufficient to sort only the head-of-queue elements from each of the flow queues. Additionally, for most of the following algorithms it is sufficient to compute a priority value when a packet reaches the head of a FIFO queue and not already on arrival at the system. Data structures for priority queues are evaluated in [24]–[27].

The following discussion is based on Weighted Fair Queuing (WFQ) alternatives. Earliest Deadline First (EDF) algorithms are not an option since they do not support fair sharing of bandwidth. As a consequence, graceful degradation of service cannot easily be implemented while maintaining in-order forwarding of packets. The smoothing effect of fair bandwidth sharing on outgoing traffic in addition avoids unsteady TCP behavior [28], decreases the accumulation of jitter, and reduces bandwidth oscillations of reactive flows in general.

2.5.1 *Weighted Fair Queuing*

The approach to adapt the behavior of a perfectly fair fluid server to the time-multiplex in packet networks is the basis for a variety of packet scheduling algorithms. The idea is introduced in [29] under the name *Fair Queueing* (FQ) and thoroughly analysed in [30]. A fluid server is configured by N positive real numbers (weights) $\Phi_1, \Phi_2, \ldots, \Phi_N$ that are assigned to N distinct FIFO queues. During any time interval $(\tau_1, \tau_2]$ when there are exactly $n \in [1, \ldots, N]$ queues backlogged the fluid server serves the n packets at the head of the corresponding queues simultaneously, each at a rate

$$r_n(t) = \frac{\Phi_n}{\sum_{j \in B(\tau_2)} \Phi_j} \cdot R(t), t \in (\tau_1, \tau_2] \qquad (1)$$

where $B(\tau_2)$ is the set of backlogged queues, which is constant in the time period $(\tau_1, \tau_2]$ and $R(t)$ is the – possibly variable – link speed, respectively. Therefore, let $W_n(\tau_1, \tau_2)$ be the amount of traffic from queue n served in interval $(\tau_1, \tau_2]$[1],

$$\frac{W_n(\tau_1, \tau_2)}{W_j(\tau_1, \tau_2)} \geq \frac{\Phi_n}{\Phi_j}, j = 1, 2, \ldots, N$$

is true for any queue n that is continuously backlogged in the interval $(\tau_1, \tau_2]$. In other words, the rate $r_n(t) = \frac{\Phi_n}{\sum_{j=1}^{N} \Phi_j} \cdot R(t)$ can be guaranteed to queue n by the fluid server and surplus bandwidth is shared fairly in proportion to the weights Φ_n of the currently backlogged queues. Without restricting the generality of the method, assuming normalized weights $\Phi_n \in (0, \ldots, 1]$ and a stable system, i.e., $\sum_{j=1}^{N} \Phi_j \leq 1$, the worst-case rate guarantee can be uncoupled from the weights of all other flows and one can state that a fluid server guarantees a rate $r_n(t) = \Phi_n \cdot R(t)$ to queue n in the worst-case independently of the behavior of all other queues.

Since packets cannot be served simultaneously in a packet system on a single outgoing link, the packetized version of a fluid server – the Weighted Fair Queuing (WFQ) server – schedules packets according to the order in which they would finish service in the fluid system. Hence, a WFQ server must emulate the fluid system in the background to function properly. In [30], an algorithm is shown that keeps track of the fluid system by maintaining a single virtual time $V(t)$. It is also shown that the packet system can only be one packet length behind the service of the ideal fluid system in the worst-case. However, there potentially is tremendous overhead for the calculation of the scheduling tags because events may appear frequently in the fluid system since all backlogged flows may finish service at the same time.

In order to find a suitable trade-off between the complexity of a scheduling algorithm, the fairness of the distribution of excess bandwidth, and the provision of sharp delay bounds, different approaches have been applied to implement WFQ.

2.5.2 *Self-clocked fair schedulers*

Self-clocked methods no longer emulate a fluid system but estimate scheduling tags by the tags of packets that are cur-

1. $W_n(\tau_1, \tau_2)$ includes service provided after time τ_1 to packets of queue n whose transmission started before time τ_1 and the service provided until time τ_2 to packets whose transmission is finished after time τ_2.

rently queued in the packet system. *Self-Clocked Fair Queueing* (SCFQ) [31] uses the finish time of the packet currently in service for the estimation of the virtual time $V(t)$ of the fluid server. *Start-time Fair Queuing* (SFQ) [32] 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) [33] and *time-shift scheduling* [34] serve packets in increasing order of their finish times. The virtual time is estimated by the minimum of the start times of backlogged flows. A second priority queue is therefore required. Self-clocked algorithms using simple estimations often only provide loose delay bounds that may depend on the number and the reservations of other flows at the scheduler.

2.5.3 Approximation by potential functions

A more systematic approach is based on the theory of *Rate-Proportional Servers* (RPS) [35]. An RPS scheduler keeps track of the state of the fluid system by a system potential function that models the progression of virtual time. A base potential function is employed to recalibrate the system potential at defined points in time. The system potential increases linearly between recalibrations – assuming the system is not idle – so as to resemble the increase of the virtual time $V(t)$. 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 [36]. SPFQ recalibrates the system potential at every packet departure. The base potential is updated at every packet arrival and set to the minimum start potential of all backlogged flows. SPFQ is similar to MSFQ. MSFQ however does not use a system potential and moreover 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) [37] uses the same recalibration intervals as SPFQ together with a simplified base potential, which does not need to maintain a second priority queue to manage start potentials.

2.5.4 Eligible packet selection

Although the amount of service by which a WFQ system may be behind a fluid system is bounded, the WFQ system schedule can be quite ahead of the fluid system [30]. This behavior will show undesired properties if feedback congestion control is used, e.g. for the regulation of best-effort traffic [38]. 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^{2}Q) [38] and its efficient implementation WF^{2}Q+ [39] sort arriving packets according to their start time in the fluid system. Only packets for which service would have been started in the fluid system are then transferred to the second priority queue, which is sorted according to finish times. WF^{2}Q+ does not need to emulate a fluid server but utilizes an approximation function similar to SPFQ and MSFQ. Opposed to these algorithms, WF^{2}Q+ considers only packets eligible for transmission. *Leap Forward Virtual Clock* (LFVC) [40] 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. WF^{2}Q and LFVC have in common that the full contents of one priority queue must possibly be copied to the other priority queue between two scheduling decisions in the worst-case.

2.5.5 Round-Robin variants

There are scheduling algorithms with low complexity that enhance the concept of a Round-Robin scheduler with virtual service ideas. *Virtual Time-based Round-Robin* (VTRR) [41] cannot provide as sharp delay or fairness bounds as schedulers which use a fluid server as reference model. Note that *Deficit Round-Robin* (DRR) [42] is often used as a comparison basis for fair schedulers and usually considered to be a WFQ implementation of low complexity. DRR can be found in many current access devices.

2.5.6 Comparing fairness

In the fluid system, from eq. (1), it immediately follows that for any two queues i, j that are continuously backlogged in the interval $(\tau_1, \tau_2]$ and have guaranteed service rates r_i and r_j respectively, the following holds:

$$\left|\frac{W_i(\tau_1,\tau_2)}{r_i} - \frac{W_j(\tau_1,\tau_2)}{r_j}\right| = 0 \quad .$$

Definition 1 (Fairness index $\mathcal{F}$ by Golestani [31]). Given any two queues i, j that are continuously backlogged in the interval $(\tau_1, \tau_2]$ and have guaranteed service rates r_i and r_j respectively, Golestani defines a fairness index $\mathcal{F}_{i,j}$ for the packet system by

$$\left|\frac{W_i(\tau_1,\tau_2)}{r_i} - \frac{W_j(\tau_1,\tau_2)}{r_j}\right| \leq \mathcal{F}_{i,j} \quad . \tag{2}$$

That is, any two queues i, j that are continuously backlogged in any interval $(\tau_1, \tau_2]$ must not receive normalized service which differs from the other queue's service by more than $\mathcal{F}_{i,j}$.

Since in the packet system a packet transmission cannot be preempted, there is a lower bound for the fairness index given by $\mathcal{F}_{i,j} \geq \frac{1}{2}(\frac{L_i}{r_i} + \frac{L_j}{r_j})$ where $L_{\{i,j\}}$ are the maximum packet lengths of the corresponding queues.

In Tab. 1 latency and fairness values for some selected scheduling algorithms are gathered from [31], [36], [37], [43]. The latency Θ_i is defined to be the worst-case latency that a maximum-sized packet of a beforehand idle flow i with guaranteed service rate r_i will experience if it arrives at an empty queue.

An undesired property of Self-Clocked Fair Queueing (SCFQ) and Deficit Round-Robin (DRR) is their dependence on the number of flows sharing the link where individual guarantees should be given. However, SCFQ achieves the best fairness of all algorithms. Assuming the same maximal packet length for all flows, MD-SCFQ realizes a better worst-case fairness than SPFQ. SPFQ in turn shows better fairness than WFQ. Since SPFQ and MD-SCFQ both belong to the class of RPS servers they have the same worst-case latency as WFQ. Obviously, one cannot have both good latency and fairness bounds. Moreover, one should always consider the

TABLE 1
Latency and fairness properties of selected schedulers. There are N flows. L_i is the maximum packet size of flow i and $L = \max_{1 \le n \le N} L_i$. The rate r_i is guaranteed to flow i by the scheduler that utilizes a link rate R. DRR is configured by assigning quantum values q_i to the flows.

Server	*Latency* Θ_i	*Fairness* $\mathcal{F}_{i,j}$, eq. (2)
WFQ	$\frac{L_i}{r_i} + \frac{L}{R}$	$\max(\frac{L_i}{r_i} + \frac{L}{r_j} + f_i, \frac{L_j}{r_j} + \frac{L}{r_i} + f_j)$ where $f_i = \min((N-1)\frac{L}{r_i}, \max_{1 \le n \le N} \frac{L_n}{r_n})$
SCFQ	$\frac{L_i}{r_i} + (N-1)\frac{L}{R}$	$\frac{L_i}{r_i} + \frac{L_j}{r_j}$
SPFQ	$\frac{L_i}{r_i} + \frac{L}{R}$	$\max(\frac{L_i}{r_i}, \frac{L_j}{r_j}) + \max_{1 \le n \le N} \frac{L_n}{r_n} + \frac{L}{R}$
MD-SCFQ	$\frac{L_i}{r_i} + \frac{L}{R}$	$\max(f_{i,j}, f_{j,i})$ where $f_{i,j} = \frac{L_i}{r_i} + \max\left(\frac{L}{r_j}, \max_{1 \le n \le N} \frac{L_n}{r_n} - \frac{r_i}{R-r_j}(\max_{1 \le n \le N} \frac{L_n}{r_n} - \frac{L_i}{r_i}) - \frac{L_i}{R}\right)$
DRR	$\frac{3 \cdot \sum_{n=1}^{N} q_n - 2q_i}{R}$	$\frac{3 \cdot \sum_{n=1}^{N} q_n}{R}$

implementation complexity in addition. WFQ must emulate the fluid system, SPFQ requires a second priority queue, and finally MD-SCFQ has to compute a base potential function.

For this case study, we select three representative schedulers, namely DRR as a common choice, SCFQ as simple approximation of WFQ with good fairness properties, and MD-SCFQ as more elaborate approximation of WFQ.

3 Modeling and Evaluation Method

In this section, the modeling and evaluation design flow followed in the case study is explained. The choice for this particular method is based on several requirements:

- Since applied early during the platform design flow, we do not want to rely on back-of-the-envelope calculations to derive system level design decisions with high impact on the overall design quality. Quantitative evaluation results are needed to rank different design alternatives.
- The evaluation must help the designer with selecting the right algorithms for implementing QoS. The algorithm models must be functional-correct. These models can in addition be used as executable (and therefore unambiguous) reference application throughout the design flow.
- The evaluation must be enabled with representative data (packet traces or traffic sources). This assumption favors evaluation by simulation so as to recognize sporadic effects and analyze the interaction of individual events.
- Certain algorithms require the modeling of time-dependent behavior. WFQ schedulers, for instance, derive priority tags for each packet dynamically at run-time that depend on the time-dependent state of service levels of each flow and the reference fluid system. The modeling formalism needs a notion of time.
- Besides the selection of algorithms, we need to model the platform hardware architecture. The hardware models must support the designer with estimations of the achievable performance if certain algorithms are implemented. Since at this point of the design flow actual hardware models in a hardware description language are not available yet and the architecture is still subject to massive change, so-called performance models of the architecture must be provided that can associate certain operations in an algorithm model with speed indicators (e.g., number of clock cycles).
- The application domain of this study is packet processing, i.e. a data flow dominant domain with stochastic, periodic, as well as sporadic events that are mainly initiated by packet arrivals and departures at the system. The processing requirements depend on the packet type and possibly the packet contents and are thus not constant. A natural choice for the model of computation is a process network, where actions are initiated by handing data (packets) from one process to the next.
- To fully explore design trade-offs in performance-cost space, the designer must be allowed to play around with different mapping decisions, i.e., how and where certain functionality should be implemented. The algorithm and architecture models must therefore be general enough at their interfaces to support variable mappings.

A comprehensive review of related evaluation methods can be found in [44]. In summary, we see domain-specific methods for signal and media processing that however lack a notion of time to model time dependent behavior [45], [46]. More general frameworks would allow the modeling of time-dependent behavior [47]. Their generality however also has drawbacks with respect to evaluation efficiency and ambiguity of implementation. Finally, selected tools have a stronger focus on paths to implementation, such as SystemC-based flows [48], [49].

3.1 Co-evaluation of Algorithm Behavior and Hardware Platform Performance

Our tool flow is implemented in the discrete event simulation and modeling framework Moses [50] that allows the evaluation of heterogeneous models of computation. The separation of algorithm and architecture (performance) models and the interaction of these models as used for our case study follows selected basic principles also implemented in Artemis [46]. An overview is shown in Fig. 3.

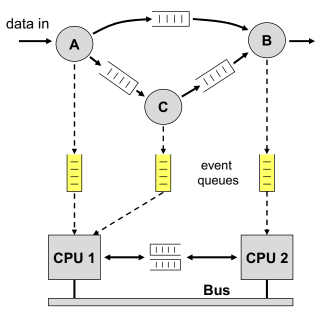
3.1.1 Algorithm model

The algorithms are described as interacting processes with a notion of time. Actions in processes are annotated with certain events that represent computational and communication requirements, as well as memory accesses. Events carry time stamps that represent packet arrival times. Processing is instantaneous at this point. Time stamps can also accommodate queuing delays due to, e.g., serialization at the bottleneck link. Events can be abstract, e.g. on the level of full tasks like the computation of a checksum or the filtering of an address, or fine-granular, describing individual arithmetic operations and word accesses. These events are collected and fed to performance models of the hardware architecture (Fig. 3 at the bottom) that associate events with processing and access delays and accumulate these delays.

3.1.2 Handling time

Compared with Artemis [46], the biggest difference is the handling of time. As motivated, we need a notion of time to consider time-dependent behavior of algorithms. As a consequence, we are not interested in the shortest schedule under full resource load as derived by Artemis, but we will see idle sections where hardware resource will not compute much since, for instance, a shaper algorithm is forced to delay packets and further processing is postponed. In order to model this interaction of algorithm and hardware models correctly, we collect events from the algorithm models periodically and analyse the computational demand during these periods. Clearly, the corresponding hardware resource is overloaded if the accumulated processing delay for the events in this period is longer than the period itself. This raises the question: What is a good period duration for the analysis of events? The longer the period, the more the resource is allowed to distribute the load during the period, i.e. peaks in the load of the resource are reduced, whereas the queuing delay of events before being processed increases accordingly. Vice versa, shorter analysis intervals mean higher resource load peaks but shorter response times. This analysis technique in addition enables the modeling of feedback from the architecture model back to the algorithm behavior, as required by, for instance, load balancing.

For the investigations in this paper we have chosen short analysis intervals since we want to focus on QoS packet processing algorithm behavior. Consequently, the resource requirements tend to be high to maintain low response times from the hardware resources. The analysis interval is one order smaller than the best-case packet delay seen in our setup. This means, when the packet delay properties will be discussed in the case study, these delays are based on processing and packet queuing delays and therefore properties of the chosen algorithms. If the analysis intervals were longer, the packet latency would additionally be affected by event backlogs at the hardware resources, subject to task schedulers on these building blocks. Incorporating task scheduling other than FCFS is straightforward in our framework but not focus of this study since computing resources are overprovisioned compared with network resources (the access link). A first approach on formalizing the fair allocation of processing and network resources together can be found in [51].



Algorithm behavior:
- Process network with notion of time to investigate time-dependent behavior
- Event annotation per basic block

Mapping:
- Periodic aggregation of events to trade-off resource load vs. processing latency
- Cloning & filtering of events to evaluate several mappings concurrently

Hardware architecture:
- Abstract, event-level performance models add up latencies for event processing
- Transformation elements allow translation of coarse-grained events to fine-granular ones

Fig. 3. Evaluation method using co-simulation of algorithm functionality and performance models of the platform architecture.

3.1.3 Mapping

At the mapping stage, where events from the algorithms are associated with performance models of the hardware architecture, we allow additional transformation elements that are introduced to ease different mapping experiments and decouple algorithm from architecture models. Usually, the algorithm designer has a good understanding of the essential operations that are needed for implementing a certain task, i.e. she/he has an idea of the number of, e.g., required accumulate and mask operations without having any specific hardware architecture in mind. On the other hand, a hardware building block is restricted to a limited set of operations and only understands certain data if presented in a certain format, as specified best by the hardware designer. A transformation element is now used to bridge both worlds. This element maps coarse-grained events from the algorithm models to fine grained operations on the hardware. In addition, certain implementation styles can be considered. A compiler-based translation may incur an additional 1.3x overhead, whereas an implementation in assembler corresponds to a 1:1 mapping, and so on. We also allow these transformation elements to clone events. In this way, several different mappings can be evaluated concurrently with a single simulation run.

3.2 Executed Operations for Algorithm Models

The choice of event abstraction on the algorithm level and the transformation to operations on the level of hardware architecture models is left to the designer. For our case study in the packet processing domain, we have chosen a relatively low abstraction level of annotations using basic blocks, as briefly introduced next. This procedure is feasible in our case, since packet processing kernels only need a few hundred lines of code. Overall event histograms can then be generated during the (simulated) run-time of the system by collecting these annotations for the executed program path. *Assumptions and limitations:* In order to generate an event histogram for packet processing at the abstraction level of a behavioral, executable algorithm specification, we make the following assumptions (which also implies a relatively low abstraction on the architecture level):

- External memory accesses caused by instruction fetches are not taken into account, i.e., the program entirely fits into the instruction cache/on-chip instruction memory.

- A set of general-purpose registers is available to hold interim results within a basic block of code and to hand interim results from basic block to basic block.[2]
- Read accesses to context information always generate RAM accesses when they appear first during the execution of an elementary packet processing task. Write accesses to context information always generate RAM accesses.
- Procedure calls are not taken into account. That means in case of a software implementation, the elementary packet processing tasks are implemented without call hierarchy, e.g., by using in-line functions.
- While counting CPU instructions it is assumed that the CPU supports register-direct, register-indirect, absolute, and indexed (register-indirect plus offset) addressing. The first three addressing modes do not generate any additional overhead to calculate the effective address. Only indexed addressing will require an address offset calculation if the offset is only known during run-time.

Our region of interest – a basic block of code – and shared data between these regions can be formalized.

Definition 2 (Basic block). A *basic block* is a code fragment given in a programming language where the control flow is only allowed to enter the sequence of instructions at the beginning of the fragment and to branch at the end of the fragment but not in between. That is, once the control flow enters a basic block, the instructions of the block are deterministically processed in order.

Definition 3 (Active variable). An *active variable* is a data item that is handed from a basic block to another basic block and read in the latter block – before it may possibly be written. An active variable is a candidate to be held by a temporary register to save memory accesses.

In the end, overall histograms of events can be gathered for arbitrary orders of arriving packets with the help of the annotations per basic block. Events will only be counted if the actual control flow passes the corresponding basic block during run-time of the simulation. In our Java implementation in Moses [50], histogram objects collect these annotations and the corresponding code is separated from the behavioral code of the algorithm.

3.3 Architecture Models

Histograms of events and further statistical data are output by algorithm models during the run-time of the simulation. This output is analyzed by models described in this subsection to derive estimations of the utilization of resources. Architecture models consider the operation-specific timing of hardware blocks. By simulating architecture and algorithm models together, the exploration of algorithm behavior and resource load is enabled without relying on stored traces.

3.3.1 CPU performance model

Only the latency introduced by a CPU's execution stage of the pipeline is taken into account by our performance models. It is assumed that the latency for other stages like fetch and decode are virtually hidden by the concurrent processing in the pipeline. If the execution stage allows variable delay for a class of operations, only the maximum value will be considered. The individual values for the event counts in a histogram generated by an algorithm model are weighted with the corresponding latencies of the execution stage and the resulting values are summed up to determine the latency of the load defined by the histogram. Since this kind of analysis is performed in fixed periods, an estimate of the average load of the CPU in a particular period can be derived. We do not make use of any superscalar architectural features of the CPU because the histograms do not provide any information about the order of the operations and we cannot perform any dependency analysis.

3.3.2 RAM timing model

Since the algorithm models can only provide access histograms without information about the order of accesses, an average latency overhead for reads and writes is considered for DRAMs that aggregates the latencies for activations and precharges. The histograms generated by the algorithm models distinguish the access type – read or write – and the length of an access. As the histograms are analyzed in regular intervals, one can estimate the load of the RAM within a period by summing up all the delays caused by the accesses in that period divided by the length of the period. The delay d caused by a single access is calculated by

$$d = \left(\left\lceil \frac{\text{access length}}{\text{memory bus width}} \right\rceil + \text{access overhead} \right) \cdot \frac{1}{\text{bus clock}} \quad .$$

3.4 Translator Models

The arithmetic part of an algorithm model can be annotated with CPU instructions in a straightforward manner as soon as data is available in registers, as described in subsection 3.2. In the domain of network processing, the data layout and organization can have a huge impact on the performance. It therefore makes sense to split these choices from the annotation of algorithm models to have a more modular design configuration space. An example in this domain is the implementation of a priority queue. Since the complexity for sorting and searching heavily depends on the chosen data structure, the behavior of priority queues has been separated from the algorithm models. The algorithm annotation only states that the algorithm wants to access the queue to, e.g., read a header field. The model of the priority queue translates this request into CPU instructions in case of a software implementation. This translation can vary greatly depending on the modeled implementation, e.g., an array versus a linked search tree. A translator model might also represent a coprocessor. In this case, no further output to architecture models is generated.

4 Case Study: QoS Packet Processing

The purpose of this section is to provide a comprehensive discussion of design trade-offs that can be evaluated on the system level for access network nodes. At this level of abstraction, the designer is mainly concerned with selecting the right functionality and assessing different design decisions, such as the mapping of functionality onto hardware

2. Note: The analysis of our algorithm models revealed that 16 registers are sufficient for the configurations presented in this paper.

resources. In the context of implementing Quality of Service (QoS) for access devices, this means that the right combination of policing, queue management, and packet scheduling must be found to guarantee defined service levels. At the same time, these algorithms must be feasible solutions, i.e. we must be able to afford their implementation complexity. In order to understand and quantify these dependencies, we evaluate the combination of different algorithm candidates and estimate the hardware resource requirements using the methods described in the preceding section.

4.1 Evaluation Setup

In the following subsections, we describe the algorithm candidates, hardware building blocks, helper elements for mapping (translator elements), traffic classes and sources (stimuli for the evaluation), and the design space of interest for our case study. The algorithm and hardware candidates have been chosen to identify corner cases of the design space and guide further design decisions.

4.1.1 Packet processing algorithms

Nested token buckets are the only policing element which is considered in the evaluation. It is assumed that packets are time-stamped with their arrival time at the packet processor and that they have passed a classification element so that a QoS class identifier is assigned to each packet.

The queue manager (QM) exchanges packets with the link scheduler component. Since a WFQ-based scheduler relies on per-flow queuing, a QM is obliged to maintain per-flow state information and a distinct FIFO-organized queue per flow must be maintained. In times of congestion a QM must warrant that only yellow-marked (non-compliant) packets suffer from loss but no green-marked (compliant) packets. We distinguish between a QM that implements fair drops of yellow packets (Fair QM). Packets are dropped from the relatively longest yellow queue with the highest overload. The overload is calculated by the ratio of the current length of a flow's queue to the size of the reserved buffer for green traffic. Since fairness is not mandatory for yellow service one can simply aggregate yellow packets from all flows and aggregate classes into a single FIFO-organized yellow queue. During congestion packets are dropped from the tail of the central yellow queue (CYQ QM). We also look at the common congestion avoidance mechanism RED [20] that we apply to the central yellow queue (CYQ-RED). Recall that the usual scope of RED assumes reactive flows. RED cannot completely avoid congestion with our settings, but we can still use our results to estimate the expense of using such a tool.

Three packet schedulers are chosen for the evaluation: Deficit Round-Robin (DRR [42]), Self-Clocked Fair Queueing (SCFQ [31]), and Minimum Delay Self-Clocked Fair Queueing (MD-SCFQ [37]). DRR is considered to have low complexity and to supply a fair distribution of bandwidth in the long-term at the expense of relatively poor delay bounds. SCFQ is based on a simple approximation of the virtual service of an underlying fluid server. SCFQ is known to provide worst-case delay bounds that depend on other flows sharing the link. Since we are especially interested in tight worst-case delay bounds, we have also chosen one representative of the class of rate-proportional servers [35] – MD-SCFQ.

4.1.2 Translator models

A priority queue is a data structure in which elements are sorted in increasing/decreasing order of assigned key values. A priority queue is needed for two tasks of the packet processing chain. The packet scheduler must sort packets according to deadlines or virtual finish times. The fair queue manager drops packets from the relatively longest queue during times of congestion. The queue manager maintains a priority queue in which flows are sorted according to their current buffer occupancies.

The right choice for a data structure for sorting and searching depends on the number of entries to sort, the type of operations to support, the characteristics of the key distribution, and the frequency of operations. A software implementation of a mature data structure is chosen, namely a heap organized binary tree [26] mapped onto a fixed-sized array. A heap shows logarithmic complexity with the number of entries to sort, inherently maintains a balanced tree, and child and parent nodes within the tree can easily be addressed by shift and increment operations if mapped onto an array. A heap competes reasonably well with more sophisticated priority queue data structures [24], [25] over a wide range of key values.

Dynamic memory allocation is a special processing block that has been separated from the algorithm models. The queue manager relies on dynamic memory allocation. A packet's payload must be stored at packet arrival. A software implementation with fixed-sized blocks of 64B is considered. The concept to store dynamically generated data of variable length as a linked list of fixed-sized segments has been established decades ago [52]. This mechanism still is the most common solution used in switches and routers since the management of dynamic memory allocation becomes simple due to the absence of fragmentation loss. It is well suited for storage systems that work under real-time constraints and where the length of the data to store may not be known at the beginning of the store process.

4.1.3 Architecture models

Two CPU timing models are distinguished to assess the load generated by packet processing tasks. ARM cores are used for a broad range of embedded systems. In particular, ARM CPUs are employed for systems where the power dissipation is a critical design constraint such as in hand-held devices. The application area of the ARM9 core we have modeled ([53]) is focused on integer computation. The ARM9 neither has a floating-point unit nor a divider unit. These operations must be emulated by integer operations. We have applied the methods described in ([54], chapter 4) to map floating-point operations and divisions to integer computations. The Hitachi 7750 ("SH4", [55]) is a more sophisticated embedded CPU with a floating-point unit including a divider. The timing values used for the evaluation are listed in Tab. 2. The timing values for floating-point operations assume a precision of 64 Bit whereas the integer operations use a precision of 32 Bit. Although the assumption of high-precision operations turns out to be inappropriate for an efficient implementation of

TABLE 2
Timing of the CPU models in clock cycles.

Operation [cycles]		*ARM9ES*	*SH4*
Integer	Min / Max / Cmp	1	1
	Add / Sub	3	1
	Multiply	5	4
	Division	224	4
	Address offset	3	1
Floating-point	Min / Max / Cmp	8	5
	Add / Sub	17	9
	Multiply	18	9
	Division	378	26
Branch		3	3
Register copy		1	1
Clock [MHz]		120	200

the algorithms, it however emphasizes worst-case corner-cases of the design space. In implementations, floating point operations are often replaced by fixed-point arithmetic.

Two representative timing models for off-chip RAMs are chosen for the evaluation. The first RAM is a common PC100-compliant SDRAM [56]. The timing employed by the SDRAM model is typical for open-page mode with interleaved address mapping. The second RAM model is a pipelined static RAM with late write. This is a typical component used for implementing off-chip caches. The used configurations are listed in Table 3.

TABLE 3
Timing of the RAM models.

Parameter		*SDRAM*	*SRAM*
width of memory bus	[Bit]	32	32
clock of the memory bus	[MHz]	100	166
read access overhead	[cycles]	4	2
write access overhead	[cycles]	3	2

4.1.4 Stimuli

In order to stimulate the packet processing chain with realistic network traffic, there are different options for choosing traffic patterns. In our case where access networks are investigated, Internet backbone traffic traces cannot be used effectively. These traces aggregate a high number of flows on a best-effort basis and their time stamps reflect backbone speeds. The integration of our access node into a real system would require to emulate the behavior of the access link provider and the contents providers, as well as a real-time emulation of our node. Finally, statistical source models are available that have been derived by analyzing traffic traces with a duration from hours up to days. Contrary to that, the periods that we use for the analysis of resource load and the behavior of the packet processing chain only cover at most some minutes so that the statistical models would not reflect a representative traffic pattern. We therefore use a set of traces generated by our own synthetic source models.

Our traces reflect an aggressive load for the packet processing chain. The following types of traces have been generated to model traffic flows on incoming LAN links, e.g. based on fast-Ethernet:

- *Constant Bit Rate (CBR) source:* Every 10 ms, a packet of the size 128 Byte is generated. In this way, a 64 KBit/s uncompressed voice source is modeled.
- *Variable Bit Rate (VBR) Video:* A packet corresponds to one video frame. MPEG or H.263 coded video shows some periodic behavior. Every eight to 12 video frames, a relatively large intra-coded frame is transmitted, whereas the other frames are predicted by some inter-frame coding and hence are smaller. The inter-arrival time of the packets is determined by the video standard (NTSC: 30 Hz frame rate, PAL: 25 Hz; double the rate for interlaced mode). The inter-arrival shows some jitter due to variable coding delay. Video traffic resembling 128 KBit/s PAL MPEG (Internet video), 42 KBit/s NTSC H.263 (video phone), and 2MBit/s MPEG2 (video-on-demand; TV broadcast) is modeled.
- *Call signalling:* Around ten packets are generated every three to four seconds with lengths varying from 128 to 512 Byte and a peak rate of 5 MBit/s to model the connection establishment procedure for voice traffic.
- *HTTP-like traffic:* HTTP-requests are modeled by bursts of five to ten packets with varying size of 40 to 300 Bytes and a peak rate of 5 MBit/s. These bursts appear every 0.5 to five seconds. HTTP-downloads are imitated by bursts of maximum-sized packets (1536 Byte) with a peak rate of up to 50 MBit/s and lengths of four to 10 ms.
- *FTP-like traffic:* FTP downloads are simply modeled by longer bursts than in the HTTP case. The burst duration varies from 10 ms to 0.5 s.
- *Transactions:* Every 0.3 to two seconds, two to five packets with varying length from 250 to 320 Byte are generated to imitate transactions with banks, bandwidth brokers, and so on. The peak rate is approx. 10 MBit/s.
- *Flooding:* Small packets flood the packet processing chain. The IP packet size varies from 40 to 46 Bytes and the peak rate from 10 to 50 MBit/s respectively. This traffic models aggressive attacks on the network processor, like denial-of-service due to ping or ARP request flooding. We will particularly use this type of traffic to check the protection of QoS flows from greedy sources and to bring congestion to the access node.

Four aggregate classes are defined containing eight flow classes all together. The WWW traffic aggregate class consists of HTTP and FTP traffic. Transactions are handled by a separate class. Video and voice form their own media aggregate class. A Virtual Private Network (VPN) aggregate class holds three classes. Each aggregate class is policed by nested token-buckets. A typical set of service level agreements with possibly several providers used for our study states:

- *WWW aggregate class:* At least 10% of the link bandwidth must be available for WWW traffic, divided into 6% for HTTP traffic and 4% for FTP traffic. If surplus bandwidth is available, WWW traffic will be allowed to occupy the whole link bandwidth. Green-only HTTP traffic should not experience longer delays than 100 ms. In the FTP case, this bound increases to 500 ms.
- *Transactions:* Transaction traffic should at least experience the service of a virtual leased line with 128 KBit/s bandwidth. It must not exceed 500 KBit/s.

- *Media aggregate class:* This aggregate class consists of voice and video traffic classes. Video packets must not experience longer delays than 40 ms. Video must at least receive 1 MBit/s of the link bandwidth. For voice, 64 KBit/s are always reserved and the delay must be below 5 ms. Media traffic is upper-bounded by 3 MBit/s.
- *VPN aggregate class:* VPN traffic, consisting of FTP, HTTP, and transactions, must not surpass half of the link bandwidth. At least 10% of the link bandwidth must be available for VPN WWW traffic, divided into 6% for HTTP traffic and 4% for FTP traffic. VPN transaction traffic should experience the service of a virtual leased line with 128 KBit/s bandwidth.

The bottleneck link is set to 10 Mbit/s. The guaranteed service levels utilize the link only by one third, whereas the maximum bounds together form overload – roughly 60% more than the bottleneck link can handle. In other words, during normal operation, traffic that passed the policer can be aggressive enough so that the access node runs into congestion and relies on congestion avoidance or recovery by the queue manager. We exploit this setup by injecting flooding traffic into one of the aggregate classes to investigate how well flows within the same aggregate class, as well as flows in other aggregate classes, are protected against misbehaving traffic. With our setup, policer, scheduler, and queue manager are challenged together. Our normal setup without the flooding sources is well below the bottleneck link speed and maximum buffer allocation of the node.

4.1.5 Design space

The main goal of our exploration of design parameters on the system level is the characterization of the algorithms with respect to memory and computation requirements. We can then determine, how many and what kind of processors are needed, how the buses should be dimensioned, whether we need additional accelerators, since programmable cores are not fast enough, and what kind and how many RAMs must be allocated.

The maximum capacity of memory required can be estimated statically. We can distinguish code, context, and buffer memory. Using our policer settings and guaranteed scheduler properties, capacity bounds can be derived by applying the network calculus [57]. Context memory requirements can be determined by static analysis of the algorithms depending on, e.g., whether per flow state is needed or not. Finally, code memory can only be determined if an actual implementation is available. The designer has to rely on her/his expertise where the complexity of the algorithm model can help as approximation of the complexity of the implementation.

As described in Section 3.3, we have implemented an exemplary set of CPU and RAM performance models to assess the allocation of different architecture building blocks. Our study focuses on these properties that we determine by simulation using the method described in Section 3.

4.2 Results and Discussion

The results for evaluating the computational requirements and QoS behavior of different combinations of policers, queue managers, and packet schedulers under varying packet processing load are displayed in Figures 4 and 5. On the horizontal direction, three different queue managers are distinguished, whereas we find three packet schedulers along the vertical axis. In each subgraph, eight traffic classes are plotted along the horizontal axis. We picked out high priority transactions (T), medium priority multimedia (MM) and virtual private network traffic (VPN), as well as low priority ftp and http traffic (WWW).

4.2.1 QoS behavior: packet latency

In Fig. 4, we have a look at the worst-case packet latency experienced by packets in each flow for a simulated run-time of 60 s. Since these values vary from a few μs for high priority voice traffic to several ms for best-effort high-volume Internet traffic, we plot these values relative to the most sophisticated combination in the upper left corner that is therefore represented by a horizontal line at value one. The setup has been chosen so that the system runs into congestion. The bottleneck link is set to 10 Mbps, whereas aggregated traffic of up to 50 Mbps enters the system.

We can recognize several interesting effects. Comparing the first row from the top with the second row, we see that the elaborate approximation MD-SCFQ of WFQ performs the same as its simple variant SCFQ with respect to packet latency. Comparing these rows with the last row, we see that no matter what queue manager is chosen, the RR-based scheduling discipline DRR always performs considerably worse than the WFQ variants. Note that the last row uses a different scale on the vertical axis than the first two rows. Comparing the first column from the left, representing the most sophisticated queue manager, with the second column, we observe that the less elaborate manager prefers one of the VPN classes and punishes the best-effort WWW classes. Superficially analyzed, this would actually be the preferred behavior. We only get the complete picture when we have a look at the loss rate (not displayed). We then see that the less sophisticated queue manager punishes the medium priority VPN flows by dropping packets at a higher rate. Consequently, more best-effort traffic is forwarded, i.e. the VPN traffic is less protected than using the manager in the first column. As a side effect, the reduced backlog of the VPN classes is forwarded with lower latency, whereas the increased best-effort volume sees higher latency.

4.2.2 Algorithm complexity and resource utilization

In order to derive design decisions, it is not sufficient to look at the algorithm behavior alone. The best algorithm is useless if we cannot implement it with reasonable and feasible resource requirements. In Figure 5, we therefore have a look at the different combinations of policing, queuing, and scheduling algorithms on a single CPU target coupled with a single RAM. The maximum resource load experienced over a simulated run-time of 60 s is shown for an analysis period length of 0.1 ms. We have a look at two CPU and two RAM alternatives to reveal corner cases. The exemplary ARM target represents an embedded core that only uses an integer ALU. The SH4 target has a longer pipeline, thus running at a higher clock rate, and also has a floating-point unit. For the RAM part, we look at widely deployed, but slower SDRAMs and expensive SRAMs as alternatives. If the 100% dotted line is crossed, we know that a single

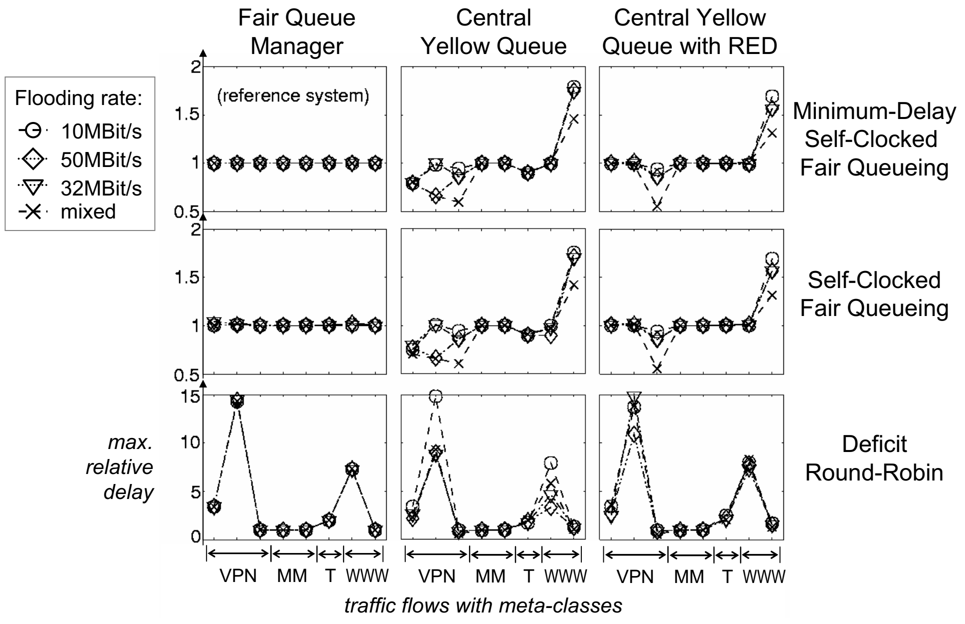


Fig. 4. Packet latency for different traffic classes and combinations of queue management and packet scheduling relative to reference system in top left corner while the system is congested. The bottleneck link supports 10 Mbps, whereas the sum of all traffic sources can be as high as 50 Mbps.

CPU/RAM combination is saturated, i.e. a multi-processor solution is needed. Each individual bar is subdivided into three pieces, representing the load required for the packet scheduler (lower part), the queue manager (middle part), and the policer (upper part). The policer is not changed in this diagram, which is why the policer bar is constant in all the nine subgraphs.

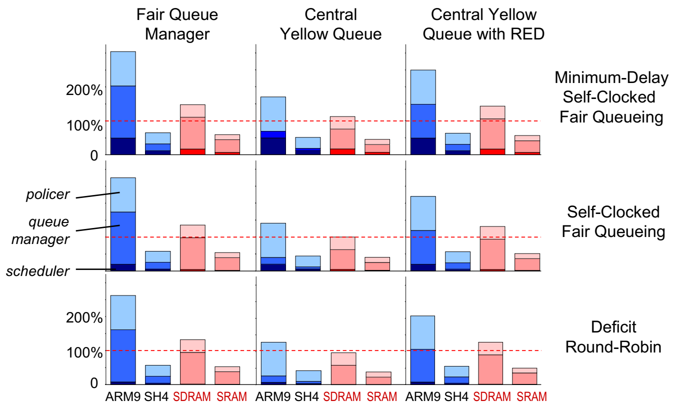


Fig. 5. Resource load for different resource types and combinations of queue management and packet scheduling for congested system. 100% corresponds to a fully utilized resource, i.e. figures above the dotted line represent overload if only a single instance is employed.

We can see that calculating fair shares of the backlog by using the queue manager in the first column from the left is expensive. Implementing RED (column on the right) is almost as expensive due to calculating average queue backlog, and so on[3]. If we can rely on the shown dual token bucket, three color policer for congestion avoidance, the implementation complexity can considerably be reduced by using a simple implementation of queue management for congestion recovery (middle column). The impact of congestion avoidance by policing versus congestion recovery by a fair (and therefore expensive) QM is underpinned in Fig. 6, where we have a look at the development in time of such a congested situation. The effect of three greedy flows is shown. All flows start at time zero for this setup. Flow 1 is bounded by the policer roughly after 1 ms before the corresponding queue is at its maximum level. Contrary to that, due to loose policer settings, the policer limits flow 2 and 3 only after 14 ms, more than 1 ms after the queue manager must start to drop packets. Within this interval when the queue manager has to cope with the incoming traffic at the peak rate, the maximum load values are experienced.

Still, several ARM processors are required for implementing the system with a less complex QM. We also see that the implementation complexity is more sensitive to the selection of the queue manager than on the choice of the packet scheduler. SCFQ and DRR can be implemented with about the same complexity in this context. Since SCFQ shows better QoS behavior as described before, we prefer SCFQ over DRR in any case. Looking at the RAM resources, we can observe that a single SDRAM is almost saturated by queue management. Since implementing queues also requires a large buffer space, using two separate RAMs, e.g. one large DRAM for queuing and one small but fast SRAM for context data, is the logical choice.

The profiling of priority queues and buffer management reveals the following: The buffer management is responsible for roughly 1.5% of the maximum CPU load and one sixth of the maximum RAM load (included in the diagrams in Fig.5). The overhead for linking segments in memory can be reduced at the expense of larger segments. The priority queues used in the fair QM and the WFQ-based link schedulers are responsible for up to one sixth of the max. CPU

3. Table-based random numbers and power-of-two approximations of exponential functions as proposed in [20] are already considered in our RED implementation.

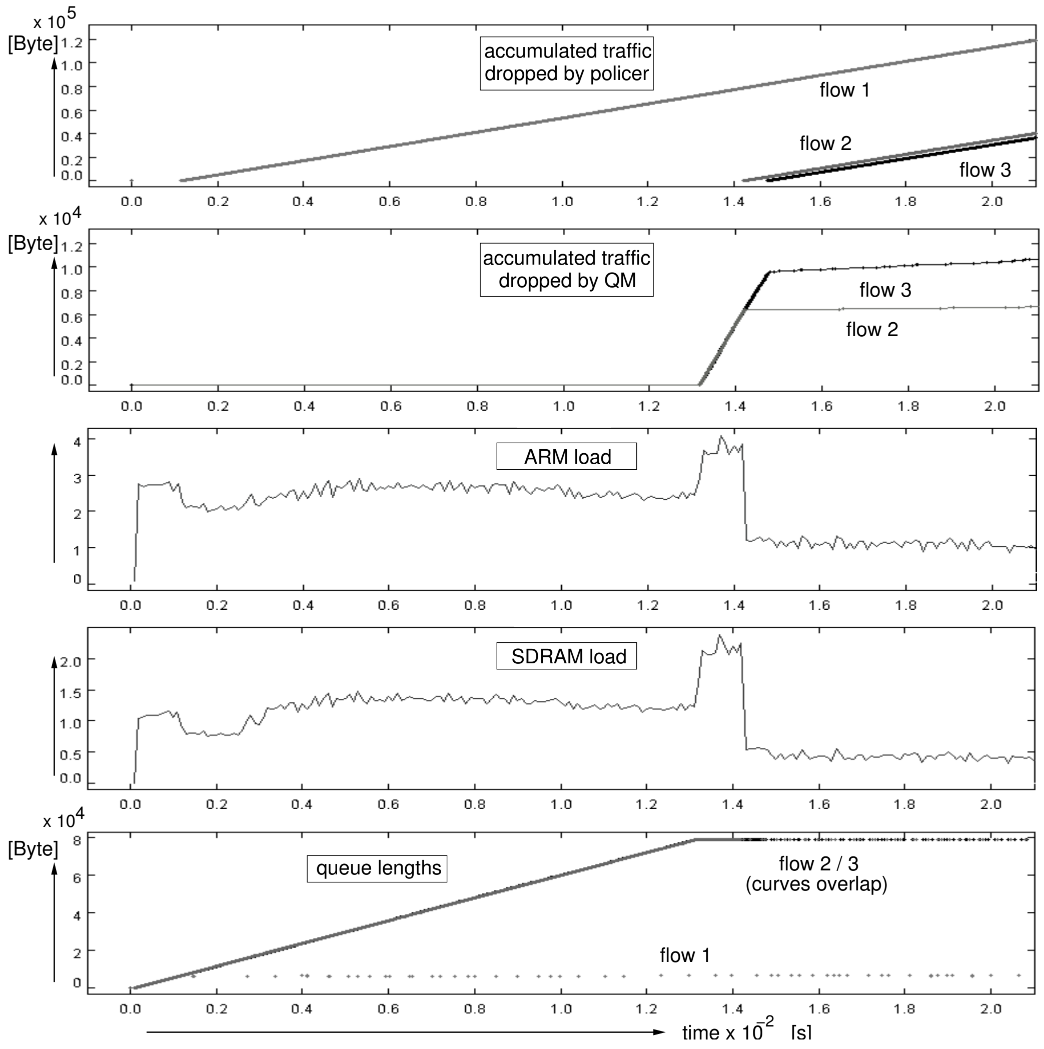


Fig. 6. Worst-case resource load situation. The system relies on congestion recovery by the QM to cope with three greedy flows.

and RAM load (also included in Fig.5), where the priority queue in the scheduler is roughly one fourth of this fraction. As a result, we are more sensitive to the selection of the right QoS algorithms than on the optimization of priority queues and buffer management.

4.2.3 A note on fairness

This subsection focuses on the fairness properties of the overall system. Better fairness is desirable for reactive flows since bursts are smoothed using fair bandwidth sharing, thus avoiding bandwidth oscillation. We use the concept of relative service as it is used for the fairness index by Golestani in Def. 1 to assess the short-time unfairness of a system. The relative service $\frac{W_i(\tau_1,\tau_2)}{r_i}$ for a flow i is defined by the served amount of flow i's traffic $W_i(\tau_1, \tau_2)$ in the interval $(\tau_1, \tau_2]$ and the reserved rate r_i.

For better visualization, we only show the relative service for five traffic flows in Figure 7. Congestion does not appear, i.e., packets need not be dropped by the queue manager. The link scheduler alone determines differences in service in this case. The relative service values are displayed for a backlog period of approx. 0.4 s for two link schedulers that represent corner cases. SCFQ shows the best fairness, whereas DRR reveals the worst. MD-SCFQ's fairness plot is very close to SCFQ's behavior and therefore not shown. The accounting resets after 0.4 s because the set of backlogged flows changes. The average slope of all curves is above one for all flows. That means, every flow at least receives its minimum reserved share of the link rate. DRR shows a variance in relative services that is three times worse than that of SCFQ. Since the fairness for WFQ-based systems only depends on the maximum packet length and the smallest reservable rate, the unfairness will not necessarily increase if more flows share the link. Opposed to that, a DRR-based system will show decreased fairness in any case if more flows are backlogged since the unfairness of DRR is directly coupled with the Round-Robin frame length (cf. Table 1).

In order to investigate the influence of the queue manager on the overall fairness of the system, we let the link run into congestion by reducing the available memory space and looked at the behavior of different queue managers. We had to recognize that the choice of the queue manager does not influence the fair distribution of service among backlogged flows by the scheduler according to the fairness index by Golestani. In fact, the queue manager may only shorten or lengthen the busy period of a flow by affecting

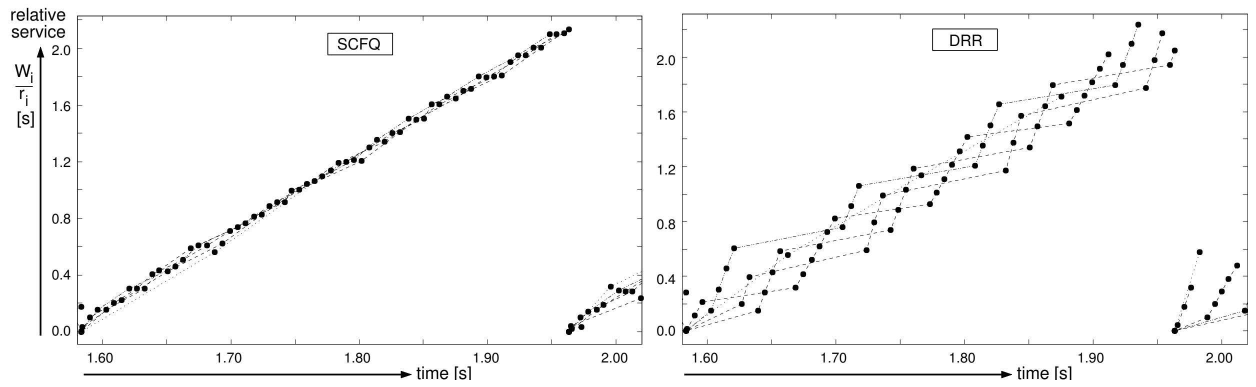


Fig. 7. Relative service for five flows backlogged for approx. 0.4 s at an uncongested link. The set of backlogged flows changes afterwards so that the accounting resets. The relative service $\frac{W_i}{r_i}$ for a flow i is defined by the served amount of flow i's traffic W_i and the reserved rate r_i.

the backlog due to loss. The decision to drop a packet in the queue manager might be unfair, in the sense that the packet does not belong to a misbehaving flow, as revealed in the preceding sections (cf. Fig.4).

4.2.4 Buffer requirements

For completeness, we summarize the memory capacity requirements derived by static analysis. For supporting up to 128 traffic flows and 32 aggregate flows, we need 14 KB of context memory, where 5 KB are needed for policing, 7 KB for scheduling, and 4 KB for queue management, respectively. The context memory for policing is mainly determined by the number of counters needed for implementing token buckets. The scheduler maintains per flow state for monitoring fairness. Finally, the queue manager monitors the backlog levels of all flows.

For maintaining packet descriptor queues (free list and fill levels) we need up to 256 KB for managing 40K entries. Accordingly, payload memory capacity is bound by 24 Mbit using 64 B segments. The sum of analytical bounds for our guaranteed traffic and SLA settings are below 1 Mbit for the described eight flows. Scaled to 128 traffic flows, 24 Mbit therefore offer enough extra buffer space for overload situations, segmentation loss, and resource sharing among aggregate flows. In summary, we recognize a memory hierarchy with three levels. The payload memory is a candidate for off-chip DRAM, the descriptor queues can be implemented on-chip in shared memory, whereas the algorithm context should be kept local to the individual processors.

In summary, we have shown how quantifying design tradeoffs early during the design flow can reveal convincing arguments for supporting design decisions that back-of-the-envelope calculations cannot reveal. We have presented our simulation-based design flow for packet processing platforms. More important than having exact numbers at hand is ranking different design alternatives correctly during this concept phase of the design. In our design flow, the sensitivity on certain design decisions can quickly be determined by playing with different mappings and translation elements that can, for instance, model different implementation efficiencies and programming styles.

5 Conclusion

We have evaluated the interplay of algorithms necessary for implementing Quality of Service (QoS) in access devices, such as CPE and DSLAMs. By combined simulation of the functionality and performance models of the hardware architecture, design trade-offs with respect to complexity of implementation and QoS behavior have been pointed out. Major results are:

- Weighted fair Queueing (WFQ) based scheduling algorithms perform considerably better than Round-Robin (RR) based versions that are most commonly used in access network setups. The difference in, for instance, worst-case packet delay by up to one order of magnitude is remarkable since only a relatively small number of traffic flows is distinguished in access devices. A simple approximation of WFQ can be implemented with about the same complexity as an RR variant in this case, while achieving better QoS properties.
- If fair scheduling is a requirement, the queuing discipline must have support for fair treatment of backlog. Otherwise, congestion recovery relying on the queue manager can heavily disturb the QoS properties of the packet scheduler.
- The investigated combinations of algorithms for policing, queuing management, and scheduling have shown that a fine adjustment of the policer (congestion avoidance) makes more sense than relying on the queue manager (congestion recovery) for coping with overload. In this way the system can rely on a simple queue management scheme and reduce implementation complexity. If policing is not an option (e.g. due to the lack of SLAs), a fair queuing manager can be implemented with about the same complexity as RED, The former option is beneficial if a high volume of UDP traffic is expected (e.g. video streams).

The described investigations have revealed the usefulness of quantitative evaluation of design alternatives early during the design flow. The sensitivity on certain algorithm choices has been quantified before determining the hardware allocation and implementation, thus avoiding costly wrong ad hoc decisions.

Acknowledgments

This work was supported in part by the Swiss Commission for Technology and Innovation (KTI), grant 4241.1, and Inalp Networks AG.

References

[1] S. Sargento, R. Valadas, J. Goncalves, H. Sousa, IP-based access networks for broadband multimedia services, IEEE Communications Magazine 41 (2) (2003) 146–154.

[2] DSL Forum, DSL evolution - architecture requirements for the support of QoS-enabled IP services, Tech. Rep. TR-059, Architecture & Transport Working Group (Sep. 2003).

[3] T. Lakshman, D. Stiliadis, High-speed policy-based packet forwarding using efficient multi-dimensional range matching, Computer Communication Review 28 (4) (1998) 203–214.

[4] A. Feldmann, S. Muthukrishnan, Tradeoffs for packet classification, in: IEEE INFOCOM 2000, Tel-Aviv, Israel, 2000.

[5] P. Gupta, N. McKeown, Dynamic algorithms with worst-case performance for packet classification, in: IFIP Networking Conference, Paris, France, 2000.

[6] P. Gupta, N. McKeown, Classifying packets with hierarchical intelligent cuttings, IEEE Micro 20 (1) (2000) 34–41.

[7] V. Srinivasan, S. Suri, G. Varghese, Packet classification using tuple space search, Computer Communication Review 29 (4) (1999) 135–146.

[8] Y. Rekhter, T. Li, An architecture for IP address allocation with CIDR, Request for Comments 1518, Internet Engineering Task Force (IETF) (Sep. 1993).

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

[10] K. Sklower, A tree-based packet routing table for Berkeley UNIX, in: USENIX Winter Conference, 1991, pp. 93–103.

[11] P. Gupta, S. Lin, N. McKeown, Routing lookups in hardware at memory access speeds, in: INFOCOM'98, Vol. 3, IEEE Computer and Communications Societies, 1998, pp. 1240–1247.

[12] M. Degermark, A. Brodnik, S. Carlsson, S. Pink, Small forwarding tables for fast routing lookups, Computer Communication Review, ACM SIGCOMM 27 (4) (1997) 3–14.

[13] S. Nilsson, G. Karlsson, IP-address lookup using LC-tries, IEEE Journal on Selected Areas in Communications 17 (6) (1999) 1083–1092.

[14] B. Lampson, V. Srinivasan, G. Varghese, IP lookups using multiway and multicolumn search, in: INFOCOM'98, Vol. 3, IEEE Computer and Communications Societies, 1998, pp. 1248–1256.

[15] H. H.-Y. Tzeng, T. Przygienda, On fast address-lookup algorithms, IEEE Journal on Selected Areas in Communications 17 (6) (1999) 1067–1082.

[16] S. Shenker, J. Wroclawski, General characterization parameters for integrated service network elements, Request for Comments 2215, Internet Engineering Task Force (IETF) (Sep. 1997).

[17] J. Heinanen, R. Guérin, A two rate three color marker, Request for Comments 2698, Internet Engineering Task Force (IETF) (Sep. 1999).

[18] B. Braden, D. D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, Recommendations on queue management and congestion avoidance in the Internet, Request for Comments 2309, Internet Engineering Task Force (IETF) (Apr. 1998).

[19] A. Romanow, S. Floyd, Dynamics of TCP traffic over ATM networks, IEEE Journal on Selected Areas in Communications 13 (4) (1995) 633–641.

[20] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking 1 (4) (1993) 397–413.

[21] D. Lin, R. Morris, Dynamics of random early detection, Computer Communication Review 27 (4) (1997) 127–137.

[22] B. Suter, T. Lakshman, D. Stiliadis, A. K. Choudhury, Buffer management schemes for supporting TCP in gigabit routers with per-flow queueing, IEEE Journal on Selected Areas in Communications 17 (6) (1999) 1159–1169.

[23] I. Cidon, R. Guérin, A. Khamisy, On protective buffer policies, IEEE/ACM Transactions on Networking 2 (3) (1994) 240–246.

[24] R. Rönngren, R. Ayani, A comparative study of parallel and sequential priority queue algorithms, ACM Transactions on Modeling and Computer Simulation 7 (2) (1997) 157–209.

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

[26] D. E. Knuth, The Art of Computer Programming: Sorting and Searching, 2nd Edition, Vol. 3, Addison-Wesley, 1998.

[27] M. Thorup, On RAM priority queues, SIAM Journal on Computing 30 (1) (2000) 86–109.

[28] C. Barakat, E. Altman, W. Dabbous, On TCP performance in a heterogeneous network: a survey, IEEE Communications Magazine 38 (1) (2000) 40–46.

[29] A. Demers, S. Keshav, S. Shenker, Analysis and simulation of a fair queueing algorithm, Internetworking: Research and Experience 1 (1) (1990) 3–26.

[30] A. K. Parekh, R. 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) (1993) 344–357.

[31] S. J. Golestani, A self-clocked fair queueing scheme for broadband applications, in: INFOCOM 94, Vol. 2, IEEE, 1994, pp. 636–646.

[32] P. Goyal, H. M. Vin, H. Cheng, Start-time fair queuing: a scheduling algorithm for integrated services packet switching networks, Computer Communication Review 26 (4) (1996) 157–168.

[33] Y.-P. Chu, E.-H. Hwang, A new packet scheduling algorithm: minimum starting-tag fair queueing, IEICE Transactions on Communications E80-B (10) (1997) 1529–1536.

[34] J. A. Cobb, M. G. Gouda, A. El-Nahas, Time-shift scheduling-fair scheduling of flows in high-speed networks, IEEE/ACM Transactions on Networking 6 (3) (1998) 274–285.

[35] D. Stiliadis, A. Varma, Rate-proportional servers: a design methodology for fair queueing algorithms, IEEE/ACM Transactions on Networking 6 (2) (1998) 164–174.

[36] D. Stiliadis, A. Varma, Efficient fair queueing algorithms for packet-switched networks, IEEE/ACM Transactions on Networking 6 (2) (1998) 175–185.

[37] F. M. Chiussi, A. Francini, Minimum-delay self-clocked fair queueing algorithm for packet-switched networks, in: Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications, Vol. 3, IEEE, 1998, pp. 1112–1121.

[38] J. C. Bennett, H. Zhang, WF^{2}Q: worst-case fair weighted fair queueing, in: IEEE INFOCOM '96, The Conference on Computer Communications, Vol. 1, 1996, pp. 120–128.

[39] J. C. R. Bennett, H. Zhang, Hierarchical packet fair queueing algorithms, IEEE/ACM Transactions on Networking 5 (5) (1997) 675–689.

[40] S. Suri, G. Varghese, G. 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.

[41] K.-H. Cho, H. 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, Vol. 3, 1998, pp. 1520–1525.

[42] M. Shreedhar, G. Varghese, Efficient fair queuing using Deficit Round-Robin, IEEE/ACM Transactions on Networking 4 (3) (1996) 375–385.

[43] D. Stiliadis, Traffic scheduling in packet-switched networks: Analysis, design, and implementation, Ph.D. thesis, Dept. of Computer Engineering, University of California, Santa Cruz (Jun. 1996).

[44] M. Gries, Methods for evaluating and covering the design space during early design development, Integration, the VLSI Journal, Elsevier 38 (2) (2004) 131–183.

[45] E. A. de Kock, G. Essink, W. J. M. Smits, P. van der Wolf, J.-Y. Brunel, W. M. Kruijtzer, P. Lieverse, K. A. Vissers, YAPI: Application modeling for signal processing systems, in: 37th Design Automation Conference (DAC), 2000, pp. 402–405.

[46] A. Pimentel, L. Hertzberger, P. Lieverse, P. van der Wolf, E. Deprettere, Exploring embedded-systems architectures with Artemis, IEEE Computer 34 (11) (2001) 57–63.

[47] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Paserone, A. Sangiovanni-Vincentelli, Metropolis: an integrated electronic system design environment, IEEE Computer 36 (4) (2003) 45–52.

[48] T. Grötker, S. Liao, G. Martin, S. Swan, System Design with SystemC, Kluwer Academic Publishers, 2002.

[49] A. Wieferink, T. Kogel, R. Leupers, G. Ascheid, H. Meyr, G. Braun, A. Nohl, A system level processor/communication co-exploration methodology for multi-processor system-on-chip platform, in: Design Automation and Test in Europe (DATE), 2004, pp. 1256–1263.
[50] R. Esser, J. W. Janneck, Moses – a tool suite for visual modelling of discrete-event systems, in: Symposium on Visual/Multimedia Approaches to Programming and Software Engineering at HCC01, 2001.
[51] Y. Zhou, H. Sethu, On achieving fairness in the joint allocation of processing and bandwidth resources: principles and algorithms, IEEE/ACM Transactions on Networking 13 (5) (2005) 1054–1067.
[52] E. Wolman, A fixed optimum cell-size for records of various lengths, Journal of the ACM 12 (1) (1965) 53–70.
[53] ARM, Ltd., ARM9E-S Technical Reference Manual, ARM DDI 0165A, *http://www.arm.com* (Dec. 1999).
[54] J. L. Hennessy, D. A. Patterson, Computer Organization & Design, the Hardware / Software Interface, Morgan Kaufmann Publishers, 1994.
[55] Hitachi Kodaira Semiconductor Co., Ltd., SH7750 Series Hardware Manual, ADE-602-124C, 4th Edition (Mar. 2000).
[56] Intel Corp., PC SDRAM Specification, Rev. 1.7 (Nov. 1999).
[57] J. Le Boudec, P. Thiran, Network Calculus - A Theory of Deterministic Queuing Systems for the Internet, LNCS 2050, Springer Verlag, 2001.
[58] M. Gries, Algorithm-architecture trade-offs in network processor design, Ph.D. dissertation, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zürich, Switzerland, Jul. 2001.