Cross-protect: implicit service differentiation and admission control

A. Kortebi, S. Oueslati and J. Roberts
France Telecom R&D

***Abstract*—**
In this paper we present Cross-protect, a combination of router mechanisms allowing quality of service differentiation while maintaining the simple user-network interface of the best effort Internet. By associating implicit flow level admission control and per-flow fair queuing in a router it is possible to distinguish streaming and elastic flows and meet their respective quality requirements without requiring specific packet marking. We describe the implied mechanisms and justify the claimed performance and scalability properties by means of simulation and analysis.

I. Introduction

Our previous work on the statistical nature of Internet traffic has led us to question the appropriateness of many proposed and standardized traffic controls. We have argued that the network should be flow-aware and have shown how implicit admission control can be used to ensure adequate quality of service for both streaming and elastic flows [10], [4]. The proposed flow-aware architecture required explicit differentiation between streaming and elastic traffic in order to control the packet delay of the former in a priority queue. The present paper builds on this prior work. We propose mechanisms allowing *implicit service differentiation*: the user-network administrative interface is that of the best effort architecture while the particular quality of service requirements of streaming and elastic flows are assured by a combination of per-flow scheduling and admission control. For reasons which are made clear later, we call the resulting architecture "Cross-protect".

The previous flow-aware networking architecture, while considerably simpler than most current propositions for realizing quality of service, shares with them two significant disadvantages. Firstly, it is necessary to control the peak rate of streaming flows by policing or shaping at the ingress imposing a hard constraint on user traffic. This constraint is unnecessary in periods of light traffic when higher flow peak rates are acceptable. Secondly, it is necessary to rely on user cooperation to realize fair bandwidth sharing between elastic flows making network performance vulnerable to user misbehaviour.

Cross-protect overcomes these disadvantages. The name derives from two features. On one hand, admission control and fair queuing are mutually beneficial: admission control ensures the scalability of the scheduling algorithm while fair queuing provides the admission conditions. On the other hand, streaming and elastic flows achieve their necessary quality of service without mutual detrimental effect.

In the next section, we present the mechanisms and functions necessary to realize Cross-protect. We present the scheduling algorithm in more detail in Section III and illustrate claimed scalability by means of analytical models. Section IV presents a number of simulation results illustrating the performance of the proposed architecture.

II. A Cross-protect router

Fig. 1 illustrates a possible division of the Cross-protect functions in an IP router. Flow identification and forwarding decisions, including admission control, are implemented in the incoming line cards. Per-flow fair queuing is performed in the outgoing line cards.

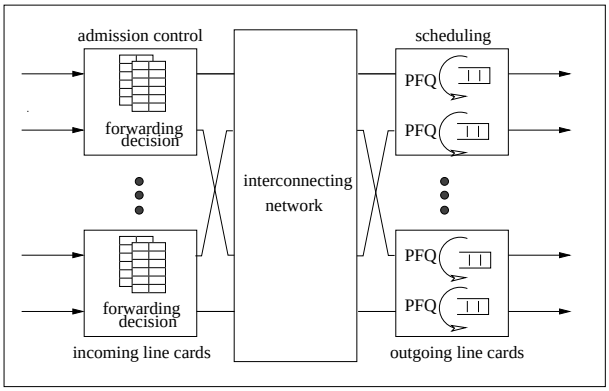

Fig. 1. A Cross-protect router

A. Flow identification

A flow is a set of packets having the same values in certain header fields among the packet address attributes. A flow is completely specified by these values associated with an idle period (or time-out). A flow is deemed to have ended when no packet is observed during this idle period.

A minimal flow identifier would be the combination of the origin and destination IP addresses. A more useful choice would be to include header fields whose value is entered freely by the user or application. Such fields include the transport port numbers in IPv4 and the flow label in IPv6.

B. Forwarding decision

In addition to the usual forwarding functions of an IP router, we propose to perform supplementary operations to decide if a packet should be forwarded or not. This decision is taken on the basis of data included in a protected flow list (PFL).

A PFL is a list of flow identifiers together with the arrival time of the last packet of each one. Flows are written to the list

depending on certain forwarding criteria. They are erased when the time since the last packet exceeds a time-out threshold.

Forwarding decisions are taken on the arrival of every packet. If the packet belongs to a protected flow, it is forwarded directly and the last packet time is updated in the PFL. If the flow is not in the list, it is necessary to proceed to a forwarding decision. The packet will be rejected if the admission conditions are not satisfied. If they are satisfied, the packet is forwarded via the designated queue. In the latter case, the flow identity may or may not then be added to the PFL, depending on some additional criteria.

One possible criterion is to apply a probabilistic decision: the flow is added with probability p; with probability $1 - p$, the packet is forwarded but the flow remains unprotected. If p is low (0.1, say), the majority of very small flows are never included in the PFL while long flows are protected with high probability after the emission of the first few packets.

The required overall size of the PFL grows in proportion to the rate of the protected link. It is necessary to dimension the memory used by the list to limit the probability of saturation. However, the only consequence of such a saturation would be that a flow would not immediately acquire the status of protected flow. Its packets would still be forwarded correctly in the absence of congestion.

C. Packet scheduling

Fair queuing scheduling ensures that link bandwidth is shared fairly without relying on the cooperative behaviour of users [7]. In association with admission control, it can also be used to guarantee the rate of admitted flows. A large number of fair queuing algorithms have been proposed in the literature. The Start-time Fair Queuing (SFQ) algorithm of Goyal et al. [8] is particularly well adapted to the present architecture.

Assume for the sake of simplicity that this algorithm realizes perfect max-min fairness with a well defined fair rate[1]. We enhance the SFQ algorithm by giving head of line priority to packets arriving to flows whose incoming rate is less than the current fair rate. Such packets are easily identified from the different parameters of the algorithm. We refer to this enhancement as Priority Fair Queuing or PFQ.

PFQ thus implicitly gives priority to the packets of streaming flows (and elastic flows) whose peak rate is less than the fair rate. Admission control maintains the fair rate above a chosen threshold allowing real time performance guarantees for a targeted category of streaming flows.

D. Admission control measurements

Admission control relies on congestion measurements performed within the PFQ scheduler. Two indicators are monitored, *fair_rate* and *priority_load* :

- *fair_rate* is an estimation of the rate currently realized by backlogged flows,
- *priority_load* is the sum of the lengths of priority packets transmitted in a certain interval divided by the duration of that interval.

Periodic measurements of *fair_rate* and *priority_load* allow a continuous monitoring of the congestion status of the link and the deduction of admission conditions to be applied to newly arriving flows.

III. The PFQ algorithm

In this section we provide a more explicit description of the PFQ algorithm and discuss a number of implementation issues.

A. PFQ data

The scheduler maintains the following data:

- a "push-in, first-out" (PIFO) queue[2] where packets are stored in decreasing order of a time stamp; each element in the PIFO has the form $\{$*packet, time_stamp*$\}$ where *packet* designates the data relating to the packet (flow identifier, size, memory location) and *time_stamp* is the packet "start tag" determined by the SFQ algorithm,
- a pointer P identifying the last of the priority packets at the head of the queue,
- a list of flows *flow_list* containing the identifier of all active flows together with a time stamp *flow_time_stamp* corresponding to the "finish tag" of the last packet of this flow, the current backlog size in bytes, *backlog*, and a counter of received bytes, *bytes*,
- a counter *virtual_time* allowing the calculation of time stamps; according to SFQ, *virtual_time* is equal to the start tag of the last packet to have begun transmission.

B. Operations on packet arrival

PFQ executes the following pseudocode on each packet arrival. The variable *packet* designates the arriving packet, L is its length in bytes and F its flow identifier.

```
1.  if PIFO congested, reject packet at head of
    longest backlog
2.  if F ∈ flow_list
3.  begin
4.     backlog(F) += L
5.     if bytes ≥ MTU
6.        push {packet,flow_time_stamp} to PIFO
7.     else begin
8.        push {packet, virtual_time} to PIFO
          behind P; update P
9.     (counter of priority bytes += L)
10.       bytes(F) += L
11.    end
12.    flow_time_stamp(F) += L
13. end
14. else begin
15.    push {packet,virtual_time} to PIFO
       behind P; update P
16.    (counter of priority bytes += L)
17.    if flow_list is not saturated
18.    begin
19.       add flow F
20.       flow_time_stamp(F) = virtual_time +L
21.       backlog(F)= L
22.       bytes(F)= L
23.    end
24. end
```

[1] Sharing is max-min fair with a fair rate θ if the rate of a fbw of peak rate d is $\min\{d, \theta\}$ and the sum of rates is equal to the minimum of the link bandwidth and the sum of the peak rates of active fbws.

[2] PIFO is shorthand for the sorting algorithms allowing packets to join the queue at any position, as determined by a time stamp, and serving always the packet at the head of the line [5]

It is first necessary to test if the queue is congested and, if so, which packet should be rejected (line 1). Different criteria for defining congestion and for choosing a packet for discard are possible. Here, we adopt the approach proposed by Suter et al. in [11].

If the flow is active, the backlog are updated (line 4). The test at line 5 is to distinguish between the first packets of a new flow with small-sized packets (lines 8-10) and the packets of a flow already having a significant backlog (line 6). Packets have priority while the cumulative volume of transmitted bytes is less than the maximum packet size, MTU. Note that this choice enables the PFQ to realize implicit differentiation between *packets* even when the switch fabric deals with smaller constant sized cells. The pointer P is necessary to realize priority queuing, distinguishing between packets of backlogged and non-backlogged flows having the same time stamp. The time stamp *flow_time_stamp* is updated (line 12).

If the flow is not active, the packet is given priority. It acquires time stamp *virtual_time* and is inserted in the PIFO at the position indicated by P (line 15). A counter keeps track of the total number of priority bytes for congestion measurement (lines 10 and 16). If the list is not already saturated, the flow is added to *flow_list* with *flow_time_stamp* equal to *virtual_time* plus the packet length L (this is the packet finish tag in SFQ) (lines 19-22).

Note that if the list is saturated, the only impact of non-insertion is that the next packet of this flow will be given priority even if it has a rate greater than the current fair rate. The flow will be identified as backlogged with high probability on a subsequent packet arrival.

C. Operations on packet departure

The pseudocode relating to a packet departure (end of emission of last byte) is as follows. The variable *packet* designates the departing packet, L its length and F its flow identifier.

```
1.  if PIFO is now empty
2.     remove all flows from flow_list
3.  else begin
4.     backlog(F) −= L
5.     serve packet at head of line
6.     next_time_stamp designates time stamp
       of this packet
7.     if next_time_stamp ≠ virtual_time
8.     begin
9.        virtual_time = next_time_stamp
10.       for all flows f ∈ flow_list
11.       begin
12.          if flow_time_stamp(f) ≤ virtual_time
13.          remove f from flow_list
14.       end
15.    end
16. end
```

When a packet leaves the queue, the operations performed depend on whether or not the PIFO is then empty. If so, it is necessary to empty *flow_list* (line 2). It is not necessary to change *virtual_time* whose value is arbitrary in an idle period. If the PIFO is not empty, we first adjust the backlog of flow F [3] (line 4). Since *virtual_time* takes the value of the time stamp of the packet that is head of line, no further operations are required if this is the same as the previous value (line 7). If not, *virtual_time* is updated (line 9) and flows which become inactive (their *flow_time_stamp* is less than or equal to the new value of *virtual_time*) are removed from *flow_list* (lines 12-13).

D. Congestion measurement

The congestion indicators *priority_load* and *fair_rate* are calculated periodically. Considering the time scales of the respective congestion phenomena, the period between two samples of *priority_load* should be several milliseconds while a longer interval of several hundred milliseconds is sufficient for *fair_rate* .

To estimate *priority_load* we maintain a counter incremented on the arrival of each priority packet by its length in bytes. Let $pb(t)$ be the value of this counter at time t, (t_1, t_2) a measurement interval (in seconds) and C the link bit rate. An estimation of priority-load is:

$$\textit{priority\_load} = \frac{(pb(t_2) - pb(t_1)) \times 8}{C(t_2 - t_1)}.$$

To estimate *fair_rate* we consider a fictitious flow emitting single byte packets and suppose these would be inserted between real packets in an order dictated by the PFQ algorithm. In a queue busy period, the number of bytes that could have been transmitted is then given directly by the evolution of *virtual_time*. In an idle period, the fictitious flow could transmit at link rate. Let $vt(t)$ be the value of *virtual_time* at time t, (t_1, t_2) a measurement interval, S the total idle time during this interval and C the link bit rate. We define the estimator:

$$\textit{fair\_rate} = \frac{\max\{S \times C, (vt(t_2) - vt(t_1)) \times 8\}}{(t_2 - t_1)}.$$

The first term is typically significant when the link is lightly loaded and corresponds to the fictitious flow using all residual link capacity. The second term is significant when the link is busy and approximately measures the throughput achieved by any real flow that is continually backlogged in the interval. Our experience suggests it is not necessary to perform more refined estimations for intermediate load conditions.

E. Scalability

The complexity of PFQ, like SFQ, is logarithmic in the number of active flows [8]. Scalability of Cross-protect is assured by the fact that this number is bounded (with high probability) by admission control. It is measured in hundreds rather than hundreds of thousands, whatever the service rate C. To see this, we successively consider three cases.

First assume all flows are bottlenecked at the link and therefore realize the fair rate. Under very general and realistic traffic assumptions, the number of flows in progress is greater than n with probability $\rho^{(n+1)}$ where ρ is the stationary link load[4] [1]. For $\rho < 0.9$, the probability of having more than 100 flows simultaneously in progress is very small ($< 3 \times 10^{-5}$). If, therefore, admission control can ensure a fair rate of at least $0.01C$,

[3] Special treatment is necessary if the flow was not included in the flow list due to saturation (line 17 of the arrival pseudo code).

[4] ρ =session arrival rate $\times$ average number of flows per-session $\times$ average flow size $/C$

the probability of blocking is negligible in normal loads. There is no advantage in relaxing this condition even when 1% of C is much larger than a reasonable target fair rate [3]. The maximum number of flows to be taken into account in this case is just 100.

Now suppose C is very large and that the load is such that no flow attains the fair rate. This is currently the case for most backbone links. Further assume, for the sake of simplicity, that packets are of constant size L. Under these assumptions the scheduler realizes FIFO queuing. No busy period of the queue involves more than one packet of any flow and every packet contributing to a busy period adds one to the length of the flow list. The list is emptied at the end of the busy period.

Assuming a large number of independent flows, the queue behaves locally like an M/D/1 system with an arrival rate equal to the sum of the packet arrival rates of all active flows. Designate this sum divided by capacity C the *local load*. The number of flows in the flow list is equal to the length (in packets) of the M/D/1 busy period. Its distribution is given in [9, p. 216]. Assuming admission control ensures the *local load* is less than 0.9, the number of flows in the flow list is less than 140 with probability 0.99.

Consider lastly a traffic mix where some flows are bottlenecked and others are not. Admission control simultaneously bounds the fair rate and the local priority load. Let the number of bottlenecked flows be N_e. Assume admission control ensures $N_e \leq 100$ and *local load* $\leq \min\{0.9, 1 - N_e/100\}$. These conditions are compatible with an objective fair rate of at least $0.01C$.

Assuming length L packets, virtual time in PFQ only takes values of the form iL for $i = 1, 2, \ldots$. A change in virtual time initiates a cycle beginning with the service of a backlogged packet and continuing until the next change in virtual time occurs. Assume *local load* is constant in a cycle. The number of flows in the flow list at the end of a cycle is then equal to the number of customers contributing to N_e consecutive busy periods of an M/D/1 queue.

We have evaluated the distribution of the flow list size for the range of possible values of N_e. It turns out that the list is largest for $N_e = 10$ and *local load* = 0.9. A saturation probability of 0.01 than corresponds to a list size of 480.

It remains to perform a more thorough performance evaluation accounting for variable packet sizes. However, the above discussion does illustrate why the required list size is independent of the service rate C and is orders of magnitude less than the number of flows in the corresponding PFL.

IV. Simulation results

In this section we present a number of ns2 simulation results to illustrate the operation of Cross-protect.

A. Simulation set-up

We simulate a bottleneck link receiving elastic and streaming flows generated according to Poisson processes. Streaming flows use UDP and consist of a succession of exponentially distributed on- and off-periods, all of mean 500 ms. The rate when on is 64 Kbit/s, packets are of length 190 bytes and flows last for 1 minute on average. Elastic flows use TCP Reno and emit packets of 1000 bytes. Their size has a truncated Pareto distribution with shape parameter 1.5, mean 25 packets, minimum 8 and maximum 1000 packets. Elastic flows count for 80% of overall traffic. The bottleneck link buffer is sized to absorb approximately one delay bandwidth product: 100 packets for a 10 Mbit/s link, 1000 for a 100 Mbit/s link. For each experiment we discard the first 100 seconds and calculate averages over the ensuing 400 seconds.

B. Behaviour in underload and overload

Most of the presented results pertain to a 10 Mbit/s bottleneck. Fig. 2 presents four sets of traces for each of 3 different configurations. The simulated configurations are, from left to right: an overall offered load of 9 Mbit/s without admission control; a load of 11 Mbit/s without admission control; a load of 11 Mbit/s with admission control. The traces depict, from top to bottom: the value of *priority_load* measured at 10 ms intervals; the value of *fair_rate* measured at 15 ms intervals; the number of flows in the protected flow list and the PFQ active flow list; the number of priority packets and the overall number of packets in the queue sampled at random instants.

The results conform to the claims of the previous sections. When load is not too close to 100% (case a), the link is virtually transparent for streaming flows. Average delay is only .48 ms and no loss occurs. Most elastic flows are too short to attain the bottleneck rate. Those that are long enough attain a throughput of around 1 Mbit/s.

In overload, without admission control, performance deteriorates progressively as the number of flows in progress steadily increases (trace 3b). Elastic flow throughput is very small as illustrated by the value of *fair_rate* (2b). Drop rate is 11%. Streaming flows become backlogged and their delay attains a maximum of 122 ms.

Admission control restores the performance of the overprovisioned link. The traces are somewhat similar in cases a and c. The admission condition used was to refuse flows if the latest measure of *fair_rate* is less than 100 Kbit/s (a condition on *priority_load* is not necessary in the considered set-up.

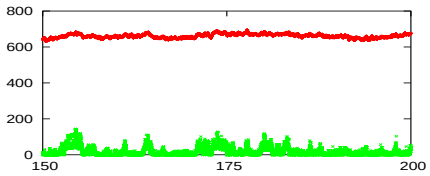

Fig. 3. Simulation results for a 100 Mbit/s bottleneck. The figure depicts the evolution of the number of flows in the PFL (red/dark) and the PFQ active flow list (green/light).

To illustrate the scalability of Cross-protect, we have simulated a 100 Mbit/s bottleneck. Fig. 3 shows the evolution of the number of flows in the protected flow list and the PFQ flow list over a 50 second interval when the offered load is 90 Mbit/s. Comparison with the third row of Fig. 2 confirms that, though the number of protected flows increases linearly with link rate, the PFQ flow list remains small.

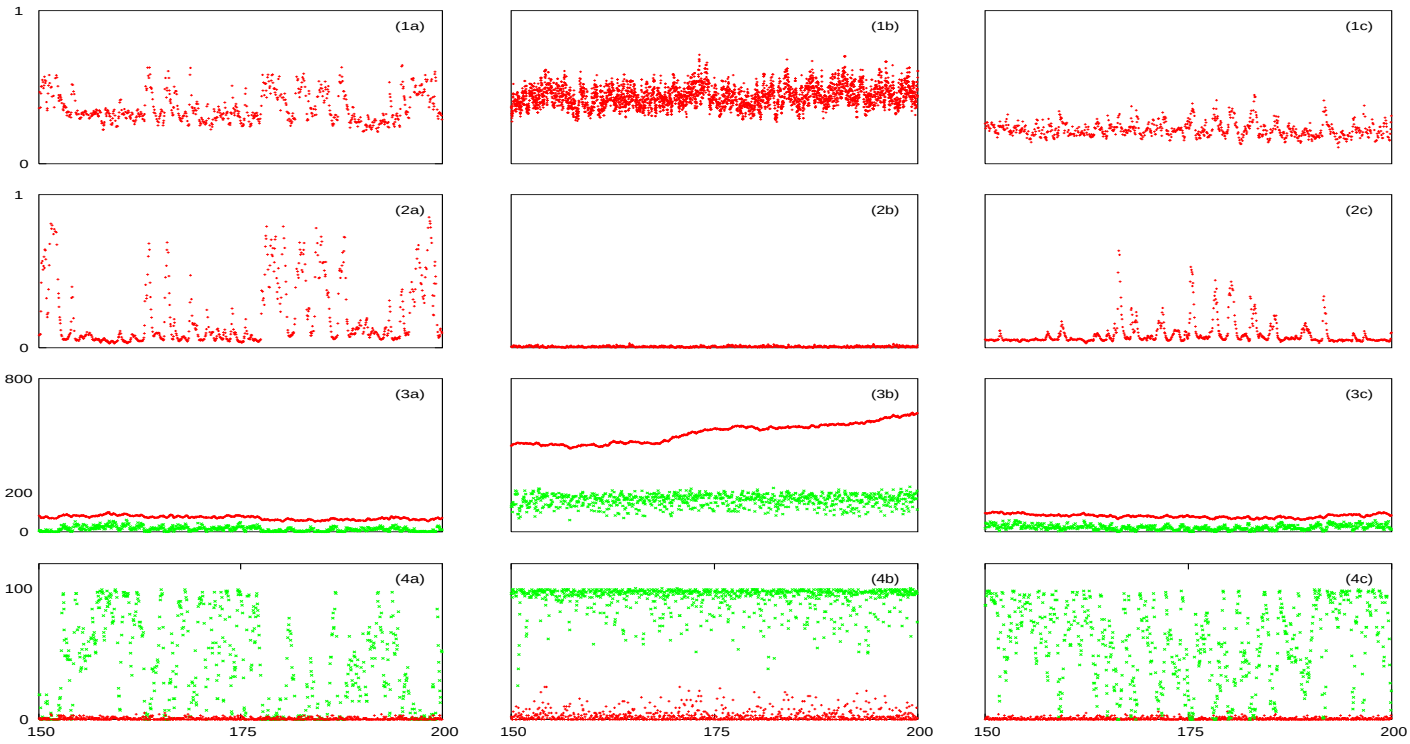

Fig. 2. Simulation results for a 10 Mbit/s bottleneck under a) normal load, b) overload without admission control, c) overload with admission control. Traces depict from top to bottom, 1) *priority_load* , 2) normalized *fair_rate* , 3) PFL size (red/dark) and PFQ flow list size (green/light), 4) priority queue length (red/dark) and overall queue length (green/light).

V. Concluding remarks

By associating the respective advantages of fair queuing and implicit admission control, Cross-protect allows differentiated quality of service guarantees while preserving the simplicity of the user-network interface of the best effort Internet.

We believe the derived flow-aware networking architecture does not introduce significant new opportunities for denial of service attacks. Indeed, per-flow scheduling is a robust service protection while flow-awareness brings new opportunities for attack detection and prevention.

Implementation of Cross-protect does require standards and can be introduced incrementally. A fully equipped AS could be made virtually transparent to quality degradation. End-to-end guarantees are, inevitably, dependent on the quality of all components of the communication path. However, the Cross-protect architecture avoids the substantial complication of class of service marking conventions and traffic conditioning presumptions.

Note finally, that flow blocking by admission control is not necessarily a negative action. It opens the possibility for adaptive routing where blocked flows can be redirected to an alternative paths. This, together with the simplicity of operating the network with a plain best effort user-network interface, implies considerable economies for the network provider.

References

[1] S. Ben Fredj, T. Bonald, A. Proutière, G. Régnié and J.W. Roberts. Statistical bandwidth sharing: A study of congestion at flow level, In Proc. of ACM SIGCOMM 2001.

[2] T. Bonald and L. Massoulié, Impact of Fairness on Internet Performance, In Proc. of SIGMETRICS 2001, Cambridge, MA, USA, June 2001.

[3] S. Ben Fredj, S. Oueslati-Boulahia, J.W. Roberts. Measurement-based Admission Control for Elastic Traffic. in J. Moreira de Souza, N. Fonseca and E.A. de Souza e Silva, Teletraffic Engineering in the Internet Era, ITC 17, Elsevier, December 2001. (http://perso.rd.francetelecom.fr/roberts/Pub/BOR01.pdf)

[4] T. Bonald, S. Oueslati, J. Roberts. IP traffic and QoS control: Towards a flow-aware architecture. In Proc. of World Telecom. Conf., Paris 2002. (http://perso.rd.francetelecom.fr/roberts/Pub/BOR02a.pdf)

[5] S. Chuang, A. Goel, N. McKeown, B. Prabhakar, Matching output queuing with a combined input/output-queued switch, in IEEE Journal in Selected Areas in Communications, Vol 17, No 6, pp 1030-1039, June 1999

[6] S. Deering, R. Hinden. RFC 2460: Internet Protocol, Version 6 (IPv6) - Specification. 1998.

[7] A. Demers, S. Keshav, S. Shenker. Analysis and simulation of a fair queueing algorithm. In SIGCOMM Symposium on Communications Architectures and Protocols, (Austin, Texas), pages 1-12, ACM, Sept. 1989. Also in Computer Communications Review, 19 (4), Sept. 1989. IEEE/ACM Transactions on Networking,, Vol. 9, No. 4, Aug 2001, pp 392-403.

[8] P. Goyal, H. Vin, H. Cheng. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. IEEE/ACM ToN, Vol 5, No 5, Oct 1997.

[9] L. Kleinrock. *Queueing Systems, Volume I*, J. Wiley, 1975.

[10] J. Roberts, S. Oueslati-Boulahia. Quality of service by flow-aware networking. In Philosophical Transactions of the Royal Society, Vol A358, pages 2197-2207, 2000. (http://perso.rd.francetelecom.fr/roberts/Pub/RO00.pdf)

[11] B. Suter, T.V. Lakshman, D. Stiliadis and A.K. Choudhury, Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-Flow Queueing, IEEE Journal in Selected Areas in Communications, August 1999.