Minimizing the Overhead in Implementing Flow-aware Networking

Abdesselem Kortebi
France Telecom, DRD/CORE/CPN
38, rue du Général Leclerc
92794 Issy les Moulineaux, France
abdesselem.kortebi@francetelecom.com

Luca Muscariello
Politecnico di Torino,
Dipartimento di Elettronica
Corso Duca degli Abruzzi 24, 10129 Torino, Italy
luca.muscariello@polito.it

Sara Oueslati
France Telecom, DRD/CORE/CPN
38, rue du Général Leclerc
92794 Issy-Moulineaux, France
sara.oueslati@francetelecom.com

James Roberts
France Telecom, DRD/CORE/CPN
38, rue du Général Leclerc
92794 Issy-Moulineaux, France
james.roberts@francetelecom.com

ABSTRACT

An enhanced flow-aware Internet is arguably a more effective means of ensuring adequate performance than implementing the complex standardized QoS architectures. This flow-aware network would provide flow-level performance guarantees for real time and data applications by implementing per-flow fair queueing and by limiting the impact of overload through flow level admission control. The paper discusses the feasibility of the implied router mechanisms and proposes original solutions that minimize the necessary overhead with respect to the current best effort network. Preferred solutions significantly reduce requirements for flow state by employing directly addressed bitmaps to record flow status, as necessary for scheduling and admission control, respectively.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Packet-switching networks*

General Terms

Performance

Keywords

Fair queueing, admission control, statistical bandwidth sharing, flow-aware networking

ANCS'05, October 26–28, 2005, Princeton, New Jersey, USA.

1. INTRODUCTION

Limitations of the current best effort datagram-based Internet architecture have led to proposals for enhanced QoS empowered architectures. Early proposals for an integrated services network with per-flow connection set up, based on RSVP signalling, have not been widely deployed for reasons of scalability [3]. The currently favoured solution is to apply class-based service differentiation [1], coupled with the traffic engineering facilities of MPLS [15], to perform traffic management based on broad flow aggregates. We have argued elsewhere that neither Intserv nor Diffserv takes proper account of the statistical nature of Internet traffic revealed by numerous measurement studies [20]. They also impose a significant cost penalty compared to the best effort network, in terms of both capital and operational expenditure.

The present paper contributes to the specification of an alternative flow-aware networking architecture [10, 19]. The objective of flow-aware networking is to augment the current Internet as sparingly as possible in order to provide necessary performance assurances for both real time and data transfer applications. According to the analysis in [20] and [10], the necessary mechanisms are per-flow fair queueing and flow level admission control. This paper discusses the implementation overhead of these mechanisms and proposes original designs that aim to minimize complexity.

A flow in the present context is a flight of datagrams, localized in time (packets are spaced by no more than a certain interval, TimeOut) and space (packets in question are observed at a particular interface) and having the same unique identifier. The identifier is deduced from header fields including IP addresses and user-specified fields like the IPv6 flow label or IPv4 port numbers. The expectation is that users define flows to correspond to a particular instance of some application such as a video stream or a document transfer.

Space locality means a given end-to-end flow has multiple instances, one at each network element on its path. To be flow-aware, these elements identify flows on the fly by examining the packet header. There is thus no requirement for

signalling and no modification to the network control plane. Minimal soft state is established, as necessary, for scheduling and admission control.

As is well known, fair queueing provides effective traffic separation, avoiding current vulnerability to users who choose not to implement congestion responsive end-to-end protocols. Fair queueing also realizes a certain implicit differentiation between real time streaming flows and bursty, potentially high rate, data transfers. This is because packet latency is limited for any flow that is not bottlenecked and this is the case for most audio and video applications.

The feasibility of per-flow fair queueing has recently been demonstrated by showing that the number of active flows (i.e., the flows that need to be known to the scheduler at any instant) is not more than several hundred, even at link loads as high as 90%, and this for any link capacity [12, 13]. The active flows are composed of a relatively small number of bottlenecked flows together with the subset of non-bottlenecked flows that currently have a packet in the buffer. In this paper, we show how complexity can be further reduced by limiting scheduling to just the bottlenecked flows, the maximum number of active flows then being limited to around 100.

If demand (flow arrival rate × average flow size) were to exceed link capacity, the number of flows to be scheduled would grow to much more than 100. To preserve scalability, and to protect the performance of flows in progress, it is necessary to limit utilization by performing flow level admission control. Since the number of flows in progress increases with link rate and could attain hundreds of thousands or more for the highest capacities, the amount of state required for admission control is considerably more than that necessary for scheduling. In the paper, we discuss two approaches for identifying the current set of in-progress flows:

1. leveraging a general purpose flow table that maintains per-flow state for a variety of applications in addition to admission control;

2. designing an *ad hoc* data structure that reduces memory requirements and complexity by exploiting the specific features of the admission control application.

The paper contains two main sections devoted to fair queueing scheduling and admission control, respectively. We base the discussion of scheduling on Deficit Round Robin [21], showing how this algorithm can be modified to identify just the bottlenecked flows whose rate needs to be controlled. The section on admission control is confined to a presentation of the necessary data structure and does not consider the measurement-based algorithm that would be implemented to determine admissibility conditions. The paper concludes with a brief discussion of the advantages of the flow-aware networking paradigm and the perspectives for implementation.

2. PER-FLOW FAIR QUEUEING

The advantages of per-flow fair queueing have long been recognized [17, 6]. As well ensuring protection against malicious use, network assured fairness frees applications from the current requirement to be "TCP-friendly". It would be possible, for example, to introduce more efficient high speed transport protocols without concern that they might be unfair to legacy TCP connections. Max-min fairness also provides low latency to the packets of flows that have a peak rate less than the current fair rate. This realizes an implicit service differentiation on recognizing that most streaming applications fall into this category. The latency of low rate flows is further reduced in the implementation described in Section 2.3.

We present the design of a reduced complexity fair queueing scheduler based on Deficit Round Robin [21, 11]. We have performed similar developments for the self-clocked time stamp based scheduler Start-time Fair Queueing [9] but omit details for the sake of conciseness.

2.1 Scalability

The huge amount of research on fair queueing since Nagle's pioneering proposal in [17] has mainly been concerned with devising schemes that realize tight fairness and delay bounds for a given set of rate controlled sources. Our focus is rather on ensuring approximate fairness for a highly dynamic population of active flows identified on the fly. The main objective is to realize fair queueing with low complexity.

Two broad classes of schedulers can be used: self-clocked fair queueing where packet emissions are ordered according to an assigned time stamp (exemplified by Start-time Fair Queueing (SFQ) [9]) and round-robin algorithms where dynamically constituted queues are visited in cyclic order (exemplified by Deficit Round Robin (DRR) [21]). Weighted fairness can be applied in the present context if the class of a flow can be determined on the fly from packet headers.

The complexity of SFQ and DRR depends on the number of flows that need to be scheduled. This in turn depends mainly on the offered link load equal to the flow arrival rate × average flow size / link capacity. For loads of up to 90%, it has been shown that with high probability this number does not exceed a few hundreds, for any link rate [12, 13, 11]. In Section 2.3 below, it is shown that the number can be further reduced to around 100.

Most flows in an IP network are short lived. The number of flows in progress at any instant on a high speed link is a variable that can attain a value measured in tens or hundreds of thousands. However, the flows that need to be accounted for by a fair queueing scheduler are considerably fewer. The scalability of fair queueing derives from the fact that most in-progress flows enter the schedule rarely, only when they have a packet to emit.

The schedule only concerns flows that have a packet in the router buffer. This fact makes per-flow scheduling scalable since the number of such flows is largely independent of the link rate and is measured only in hundreds [10, 12, 13, 11].

Some of the flows to be scheduled are bottlenecked in that they could attain a higher rate if the link in question had unlimited capacity. Traffic models predict that the number of bottlenecked flows in progress is less than 100 with high probability as long as link load is not higher than 90%.

Most flows in progress at any instant are not bottlenecked. Their rate is limited by other constraints on their path (access links, notably) to a peak value less than the fair rate offered by the link. Only the bottlenecked flows strictly need scheduling. We explore how DRR can be modified to separate out the other flows leading to a less onerous implementation.

```
1.  on arrival of l-byte packet p of flow f:
2.  if f ∈ ActiveList do
3.    update FIFO.f
4.    Queue.f += l
5.  else
6.    add f to ActiveList
7.    initialize FIFO.f
8.    Queue.f = l
9.    Deficit.f = 0
10.   add f at end of DRR schedule

11. transmit packets in DRR schedule order

12. when flow f is scheduled:
13. Deficit.f += Quantum.f
14. while (Queue.f > 0)
15.   get head packet from FIFO.f; l = packet size
16.   if (l > Deficit.f) skip while loop
17.   else
18.       emit packet
19.       Queue.f -= l
20.       Deficit.f -= l
21. if (Queue.f = 0) remove f from ActiveList
```

Figure 1: Pseudocode of DRR with a dynamic list of active flows.

2.2 DRR scheduling for a dynamic set of active flows

Flows that need to be accounted for by the DRR scheduler are logged in a table called ActiveList. Necessary per-flow state for a given flow f using DRR is as follows:

- Identifier.f - the flow identifier (possibly a hash of the relevant header fields)
- Queue.f - current length in bytes of flow f queue
- Quantum.f - value of flow f quantum ($\geq$ MTU bytes)
- Deficit.f - current flow deficit
- FIFO.f - addresses of head and tail packets of a linked list forming the flow f FIFO
- Next.f - the next flow in the DRR schedule following flow f.

Succinct pseudo-code for DRR is given in Figure 1. Flows are temporarily logged in ActiveList when they first emit a packet after a period of inactivity and remain there until they have no packets to emit at the end of one of their scheduled service quanta. A given in-progress flow may enter and leave ActiveList several times during its lifetime. In particular, non-bottlenecked flows enter and leave this structure once for every packet.

The ActiveList data for one flow in a typical realization amounts to some 16 bytes of memory. The overall memory requirement depends on the maximum number of flows that need to be recorded at any time. To limit this requirement we propose to add an additional data structure and modify the scheduling algorithm to distinguish between bottlenecked and non-bottlenecked flows, as described in the next section. Since ActiveList can then be limited to a capacity of around 100 flows (see below), it is feasible to use content addressable memory allowing rapid consultation of flow status in line 2 of Figure 1.

```
1.  on arrival of l-byte packet p of flow f:
2.  if f ∈ ActiveList
3.    update FIFO.f
4.    Queue.f+ = l
5.  else
6.    compute the NewFlows address i of flow f
7.    if (Bytes.i = 0)
8.       Bytes.i = 1
9.       send packet p to priority queue
10.   else
11.      add f to ActiveList
12.      initialize FIFO.f
13.      Queue.f = l
14.      Deficit.f = 0
15.      add f at end of DRR schedule
```

Figure 2: Modified DRR packet enqueueing pseudocode using NewFlows.

2.3 Identifying bottlenecked flows

We aim to identify non-bottlenecked flows and avoid inserting their packets in the DRR schedule. In addition, we seek to emit the packets of these flows with priority, ahead of the packets of bottlenecked flows. As explained in [10] and [11], this allows a form of implicit service differentiation since streaming flows typically have a peak rate less than the fair rate and thus naturally fall into the non-bottlenecked category. Their packets are forwarded with very low latency. The maximum flow rate for which low latency is assured can be engineered by choosing admission control thresholds that maintain the fair rate high enough, even under overload.

To identify bottlenecked flows, we use an additional data structure called NewFlows. NewFlows is an array of M words of b bits. For each flow f there corresponds a unique NewFlows word i, determined by a hash function applied to the flow identifier. The word content Bytes.i counts the number of bytes emitted by flows that map to that address. We assume the NewFlows address of a given flow is distributed uniformly between 1 and M.

All NewFlows words are reset to zero at particular instants (defined below) such that the count for a flow would exceed a certain threshold between two resets if that flow were bottlenecked. Thus, with a certain degree of imprecision, any packet that does not lead to overflow is considered to belong to a non-bottlenecked flow and is sent to the priority queue.

It is possible to perform the above operations using a flow dependent quantum when defining the bottlenecked status (i.e., when DRR is used to perform weighted sharing). In the absence of permanent flow state, this weight would need to be derived from packet header fields determining the flow class of service. A flow mapping to word i would be deemed

bottlenecked on a packet arrival if the packet size added to the current value of Bytes.i exceeds the flow quantum.

The choice of word size b determines the precision of the bottlenecked/non-bottlenecked classification. To maximize precision for a given b, the increment for a packet of length l would be $\lceil (2^b - 1)l/\text{Quantum} \rceil$.

In fact, the simplest option of setting $b = 1$ works satisfactorily on the traces we have tested. NewFlows is then simply a bitmap and any flow, whatever its quantum, is deemed to be bottlenecked and included in ActiveList if more than one packet arrive between two resets. The modification to the pseudo-code for DRR packet enqueueing with this choice of b is shown in Figure 2. Note that this choice of b facilitates a hardware implementation avoiding the potential complexity of the reset to zero operation.

Packets sent to the priority queue are served until the queue empties. Service then passes to the ActiveList flow that is currently at the head of the DRR schedule. This flow emits its quantum according to lines 12 to 21 in Figure 1. Service returns to the priority queue if necessary between the quanta of two successive bottlenecked flows.

NewFlows is reset at times t_n, $n = 1, 2, \ldots$ Epoch t_{n+1} is determined iteratively as follows:

$$t_{n+1} = t_n + 8\text{MTU}/\text{FairRate}(t_n)$$

where $\text{FairRate}(t_n)$ is an estimate of the fair rate in bits/s currently realized by the DRR scheduler. By fair rate, we mean the rate that would currently be realized by a bottlenecked flow with the minimal quantum of MTU bytes.

The fair rate estimate can be realized in several ways. In our experiments, we evaluate the rate a fictitious bottlenecked flow could have achieved in successive constant length intervals and perform an exponentially weighted average of these rates. The choice of interval length and smoothing parameter settings is not highly critical. It is preferable to average out random fluctuations that occur on the scale of the DRR cycle time while remaining responsive to significant changes in the flow make up. For the results reported below we have used an interval of 100ms and a smoothing parameter of 0.9 (i.e., new average = 0.9 × old average + 0.1 × new measure).

2.4 Accuracy

The identification of bottlenecked flows is necessarily imperfect. The above algorithm introduces both false positives (a flow is wrongly supposed to be bottlenecked) and false negatives (a bottlenecked flow is not detected as such).

False positive errors occur in the following cases:

- a small word size b leads to an overestimation of the number of emitted bytes;
- two or more flows map to the same NewFlows address and jointly lead to overflow.

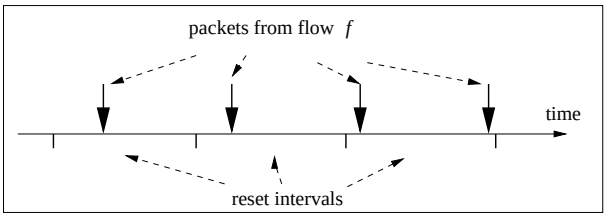
The probability of the first type of error depends on traffic characteristics and the relative quanta of different flows. For trace data we have tested, a single bit is almost as effective as 8 when all quanta are equal (see Section 2.5). A greater value may be necessary in weighted sharing if it is necessary to preserve low packet latency for flows doted with a relatively large quantum.

The probability of the second type of error can be controlled by the choice of the number of elements in the array NewFlows. Consider a bitmap realization ($b = 1$) and assume n packets arrive from distinct flows in some NewFlows reset interval. These arrivals give rise to x false positives if their images map into exactly $n - x$ elements. The probability of this event can be derived from classical results on occupancy theory [8, page 102]. The number of false positives has approximately a Poisson distribution of mean $n^2/2M$.

We discuss typical values of n is Section 2.5 with respect to trace data. It turns out that a small bitmap of a few thousand bits is sufficient to limit the rate of false positives to a fraction of 1%. The memory requirement for a given rate of false positives can of course be minimized by using Bloom filters [2] or multistage filters [7], at the expense of calculating several addresses for each flow identity.

Note that the consequence of a false positive on perceived performance is slight. The impact on bottlenecked flows is negligible since their packet latency is slightly improved. Only the latency of wrongly classified non-bottlenecked flow packets is greater that what it would have been in the priority queue. However, this latency is not high (one DRR round) and still low enough for most applications.

False negative errors can occur for a flow whose incoming rate is slightly larger than the fair rate. This is illustrated in Figure 3 where the flow is only recognized as being bottlenecked when two packets arrive in the same reset interval. Clearly, only flows whose rate is marginally greater than the fair rate are affected by this kind of error. The impact is to extend somewhat the grey area where flows are sometimes bottlenecked and sometimes not because of fluctuations in the fair rate. This imprecision occurs naturally due to the statistical nature of traffic, whatever the scheduling algorithm.



The flow has a rate slightly greater than the fair rate but is only detected when it emits two packets in the third reset interval.

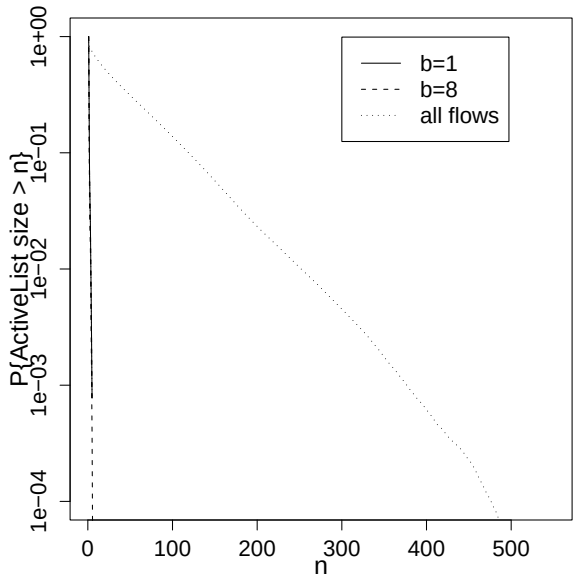
Figure 3: Delayed detection of a bottlenecked flow ($b = 1$)

2.5 Performance

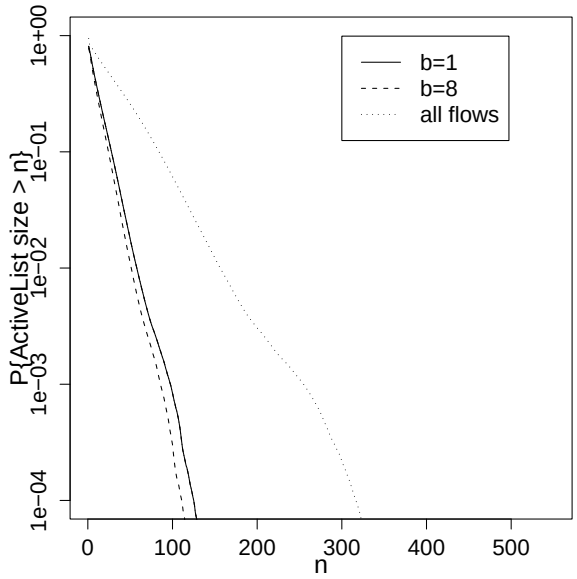
To evaluate the effectiveness of the NewFlows structure in reducing the number of flows to be scheduled we have performed simulations using trace data with flows defined by the usual 5-tuple. Two traces are used:

- *ADSL*: five minutes of traffic recorded on a backhaul link concentrating the traffic of a large group of ADSL users: link rate 155 Mbps, utilization 28%, total number of flows 8.5×10^5;
- *Abilene*: five minutes of traffic recorded on the Abilene link between Indianapolis and Kansas: link rate 2.5 Gbps, utilization 13%, total flows 2.3×10^6.

We input the trace data to links of reduced capacity to artificially augment utilization. Figure 4 plots the complementary distribution of the number of flows in the DRR ActiveList for the two traces at load 90% for three configurations: without NewFlows, with a NewFlows structure with one byte words ($b = 8$), with a NewFlows bitmap ($b = 1$). All flows have the same quantum of MTU bytes.



(a) *ADSL*



(b) *Abilene*

Figure 4: Complementary distribution of ActiveList size from trace driven simulations at utilizations 90%

The results show a dramatic reduction in ActiveList size for ADSL traffic. This is due to the fact that the peak rate of all flows is limited by their access line to less than 1 Mbps. No flow is bottlenecked in this case (and FIFO queueing would have been sufficient!). The reduction for the Abilene trace is roughly by a factor of 3. The bitmap works almost as well as an array of one-byte counters.

The above results were obtained with a NewFlows dimension $M = 16000$ resulting in a negligible proportion of false positives. Similar distributions are obtained with $M = 1000$, the rate of false positives increasing to around 5×10^{-3} for both traces. The formula in Section 2.4 predicts a rate per reset interval of $n/2M$ where n is the number of arriving packets. In the simulations, for both traces, the mean value of n is between 10 and 20 with rare peaks of 100. The formula thus predicts the empirical result.

The value of n depends on the arrival rate of packets from non-bottlenecked flows and the length of the reset interval. Using the queueing models presented in [13], the average fair rate at 90% link utilization should not be less than $0.1C$ giving an average reset interval of at most MTU$/0.1C$. Assuming a mean size of non-bottlenecked flow packets of γMTU ($\gamma = 0.3$, say), the packet arrival rate is less than $0.9C/\gamma$MTU. The product of these two terms is $9/\gamma$ (or 27 if $\gamma = 0.3$).

Mean packet delay for flows handled by the scheduler is 0.82 ms for the Abilene trace (the transmission time of an MTU-sized packet is 33 μs). Packets sent with priority have a mean latency of only 0.05 ms. The penalty of a false positive for a non-bottlenecked is thus very small in this example.

By dimensioning ActiveList for a maximum of 100 flows, say, it will occasionally occur that a new flow should be added when the table is already full (with probability 10^{-3} for the Abilene trace). This flow might be allowed to continue transmitting its packets with priority (at the risk of longer latency for competing flows) or be constrained to share quanta with other bottlenecked flows. One approach would be to add the packets of such flows to the ActiveList flow that is currently last in the DRR cycle.

3. PER-FLOW ADMISSION CONTROL

Per-flow fair queueing is feasible, as explained above, as long as link load can be controlled. In overload, when demand exceeds capacity, congestion is manifested by an increasing number of active flows and a decreasing fair rate. In addition to significant quality degradation for all flows, this would clearly stress the performance of the scheduler. To avoid this situation, the flow-aware network applies admission control at flow level. In this section we discuss the data structures necessary to distinguish the packets of new flows from those belonging to flows already in progress. The measurement-based algorithms used to determine when a new flow should be rejected are beyond present scope.

3.1 Implicit admission control

Based on measurements of the current level of congestion provided by the scheduler (estimated fair rate, link load), admission control is used as necessary to ensure that the performance of all in-progress flows is protected. This is generally only necessary in case of overload, occurring in situations of failure, for example. Flow level admission control can be performed without the need for explicit signalling or resource reservation.

Though identification of SYN and SYNACK packets has been used successfully to perform implicit admission control for TCP connections [14, 16], this solution cannot be genera-

lized to the present definition of flow (see Introduction). To detect a new flow it is necessary to maintain a data structure that records the identity of the flows currently in progress, i.e., those having last emitted a packet on the considered link within the time out interval TimeOut. A packet belongs to a new flow if its identifier is not already present in this structure.

If the link is currently in congestion, the packet of any new flow is simply discarded. The discard of the first packet, or packets, of a flow is the *implicit* signal to the user that the link is congested. We expect applications to be designed to react appropriately to this implicit signalling, by emitting probe packets during a connection establishment phase, for example. In the absence of congestion, the packet is forwarded and the flow added to the protected flow list.

The protected flow list is soft state with entries updated on the arrival of a packet from the corresponding flow. State expires when no packet is observed in an interval of length TimeOut. The appropriate value of TimeOut is a compromise between the need to limit memory requirements and the desire to avoid interrupting flows that can naturally have a long inter-packet interval. A choice of around 2 seconds may be appropriate for admission control bearing in mind that applications like conversational voice can be designed to emit keep-alive packets to maintain protection during prolonged silences.

Flow blocking is clearly un undesirable outcome for any affected user and may not always be preferred to the low level of performance that would result if admission control were not applied. However, it should be noted that admission control is a measure performed only in exceptional overloads and arguably leads to a state that is globally preferable to generalized congestion.

The case for admission control is reinforced by the observation that admission refusal on some link does not necessarily imply definitive blocking. It may be possible to route the flow over an alternative path by applying some form of load sensitive routing. Admission refusal on the first tested path is the necessary signal that an alternative path should be used. The potential for realizing load sensitive routing by exploiting the admission control mechanism is a further advantage of flow-aware networking.

3.2 Detection of new flows

It is important to note that admission control tolerates a certain imprecision in the detection of new flows. The objective of admission control is to protect the quality of service of on-going flows by reducing the arrival rate of new flows in times of congestion. Given the self-correcting properties of measurement-based admission control, it is generally sufficient that a reasonably high proportion of flows be rejected. Congestion will be relieved as long as the departure rate of flows in progress remains greater than the residual arrival rate when admission control is (imperfectly) applied. It is important on the other hand that no in-progress flow be falsely identified as a new flow. This could lead to interruption of the flow in question in the event of congestion.

The size of the required data structure is clearly a concern for reasons of scalability. Note, however, that this structure is local to the interface on which admission control is performed and only needs to record a subset of flows handled by the router. It is possible to reduce the required size by further partitioning flows into independent structures. For example, a table of flows in progress might be maintained for each combination of incoming and outgoing interfaces. This possibility notably facilitates the search operation necessary to determine if a given flow is currently in progress.

3.3 A table of flow identifiers

The obvious approach is to maintain a table containing a list of flow identifiers together with the epoch of the last packet to have been emitted.

A table of flows in progress may have other uses than just admission control, including service accounting, traffic regulation and policy routing, as discussed by Xu and Singhal [22]. These applications motivate the design of flow table architectures capable of holding data for several million flows with a combined bit rate of 100 Gbps. These data obviously include more than just the last packet epoch in order to meet the requirements of the considered applications. A major challenge in such designs, related to the general problem of garbage collection, is to purge expired flows from the structure.

One particularly interesting use of the flow table is as a route cache. The outgoing interface for a given flow only needs to be determined for the first flow packet. All subsequent packets read this interface from the flow table. As well as improving forwarding efficiency, route caching can be used to perform adaptive flow routing: if admission control detects congestion on one route, an alternative can be chosen allowing the flow to proceed.

Caspian Networks actually implement a flow table in their router [4]. The table implemented on OC 192 (10 Gbit/s) line cards is capable of storing some 6 million flows with a flow arrival rate of 2 million flows/sec. Admission control and route caching are among the applications implemented by Caspian. NetFlow [18] and similar router software use flow tables for multiple purposes but are not designed to be used for admission control.

While the implementation of a flow table may be costly (in terms of required memory and the complexity of operations in the fast path), its multiple potential uses might justify the investment providing leverage for our admission control application. In case this investment is not forthcoming, we have sought and defined a simpler *ad hoc* data structure, as described next.

3.4 A directly addressed protected flow list

Finding the appropriate entry in a flow table typically relies on a combination of direct addressing and searching. A hash of the flow identifier locates a section of the memory and an intelligent search is then conducted among the flows mapping to this section [22]. In the present approach, addressing is entirely direct: a hash of the flow identifier determines the "register" of a particular flow. For this solution to be economical, the register contains just the data necessary to determine whether the flow is in progress of not. We propose two realizations, one using a bitmap, the second an array of words. Both avoid the costly requirement to purge expired flows. This structure also exploits the tolerance to imprecision of the admission control application mentioned in Section 3.2.

3.4.1 Bitmap

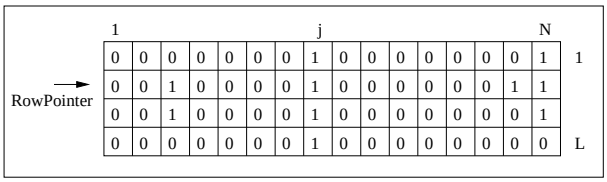
The in-progress status of a flow can be specified by a single bit: the bit is set to 1 when the flow in question is in

progress. To account for the time dimension we form a two-dimensional array of bits. The number of columns is the number N of possible values returned by the flow identifier hash function. The number of rows is a system parameter L.

The bit map represented in Figure 5 is updated as follows:

1. on a packet arrival, determine the flow address j and set all bits in column j to 1;
2. at times $i\tau$, set the bits of row i modulo(L) to zero.

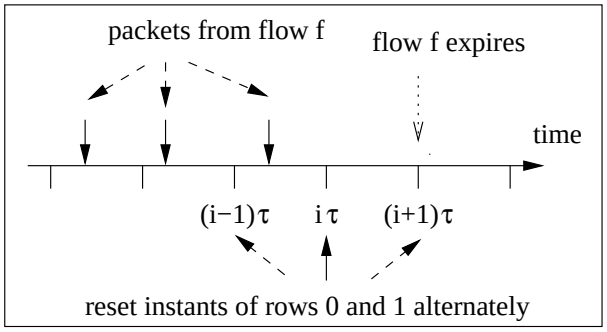
A flow mapping to column j is deemed not in progress if all bits in this column are zero. Step 1 is performed for in-progress flows and any new flow that is accepted by admission control.



All bits of column j are set to 1, indicating that a packet with a flow identifier mapping to j has arrived since the last row reset action. The variable RowPointer points on the next row to reset.

Figure 5: Bitmap representation

The choice of τ and L determine the value and precision of the time out used to determine that a flow has expired. A flow expires time T after the last packet arrival such that $(L-1)\tau < T \leq L\tau$. We can thus choose $(L-1)\tau = \text{TimeOut}$, the flow time out. The cost of imprecision determined by the granularity L is the need to record flows longer than is strictly necessary, by $\tau/2$ on average (see Fig. 6).



In this example L=2, hence τ = TimeOut. The last packet of the flow f arrives in the interval $(i-1)\tau, i\tau$, and the flow actually expires at reset instant $(i+1)\tau$. The imprecision in detecting the expiration of a flow depends on the arrival instant of its last packet in the current reset interval.

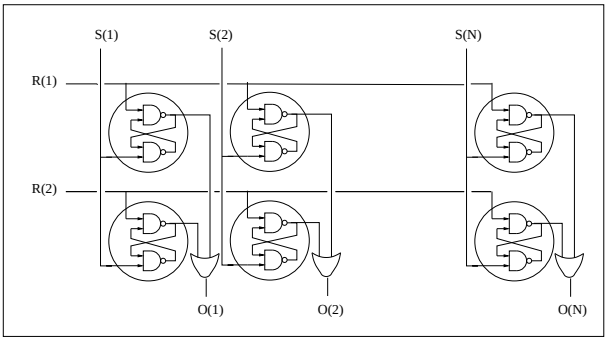
Figure 6: Delayed detection of an expired flow

A false positive error occurs when the first packet of a new flow that should be blocked is wrongly assimilated to a flow in progress. This will occur when the new flow maps to the same address as another flow currently in progress. Assuming the hash function maps flows to addresses uniformly, the rate of false positives is E[FlowsInProgress]/N and can be controlled by the choice of N.

As discussed in Section 3.2, a reasonably high rate of false positives, of 10% say, is not harmful for measurement-based admission control. Note, on the other hand, that there are no false negatives: any flow mapping to a column of zeros is certainly a new flow. This is important since no in-progress flow will then be wrongly interrupted by admission control.

In fact, only two rows of bits are required. By Little's rule, for a given rate of false positives, E[FlowsInProgress]/N, N must be set in proportion to the average flow sojourn time (FlowDuration + TimeOut + $\tau/2$). Given that $\tau = \text{TimeOut}/(L-1)$, it is easy to verify that NL, the overall size of the bitmap, is minimized in all cases when $L = 2$.

A possible realization of this bitmap in terms of logic gates is shown in Figure 7.



Two latch circuits in column i are set to 1 by a pulse on $S(i)$ on a packet arrival; each row of circuits is reset to 0 alternately by signals on $R(1)$ and $R(2)$ at intervals of TimeOut; output $O(i)$ is 0 when no flow mapping to i is in progress (the time since the last packet is between TimeOut and 2 × TimeOut).

Figure 7: Realizing the bitmap of flows in progress

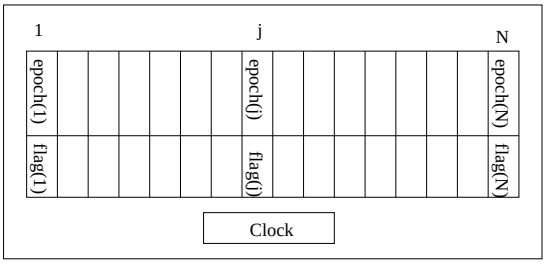
3.4.2 An array of words

An alternative structure better suited to realization in software is an array of N words each of b bits. Word j contains the epoch of the last arrival of a packet whose flow identifier maps to address j. The flow is not in progress if the difference between this time and the current clock value is greater than M. The clock is a counter coded on b bits that is incremented every σ seconds.

The choice of M and σ determine the value and precision of the time out used to determine flow expiration. A flow will expire after time T such that $(M-1)\sigma < T \leq M\sigma$. Thus M and σ have the same relation to TimeOut as L and τ in the bitmap of Section 3.4.1.

In addition to false positives due to two flows mapping to the same address, it is possible to wrongly classify a flow as in-progress if the time since the last packet is between 2^bK and $2^bK + M$ for some integer K. Assuming the flow and clock values are independent, the false positive probability is $M/2^b$ (e.g., an error rate of 3% for $b = M = 8$). To prevent a malicious user seeking to profit from this type of error by emitting successive packets until the timing condition is met, it would be sufficient to indicate by a flag bit that the stored time is out of date whenever at least one packet map-

ping to the corresponding word has been rejected. Figure 8 illustrates this data structure.



Each register is composed of an epoch coded on b bits and a one bit flag.

Figure 8: Array of words representation

3.4.3 Load sensitive routing

The above structures do not allow route caching. To perform load sensitive routing without this, it would be possible to proceed as follows. Perform load balancing by choosing between two or more links to forward a packet using a hash function applied to the flow identifier. This identifier includes a user chosen field such as the IPv6 flow label or transport protocol port numbers. If the chosen link is congested, the application on noting packet discard could reattempt using a different flow identifier (i.e., by changing the user chosen field). Load balancing thus enables a randomized route selection and can be exploited by applications to avoid blocking on a congested link. It clearly remains to more fully define this approach to load sensitive, flow by flow routing.

3.5 Randomized protection

We can further exploit the error tolerance of admission control by only protecting new flows with a certain probability p. Thus, for instance, a fraction $(1-p)$ of single packet flows will not give rise to an entry in the flow table or directly addressed structure. On the other hand, long flows will be protected after emitting $(1-p)/p$ packets on average.

Given the heavy-tailed nature of flow size and duration [23], a choice of $p = 0.1$, say, would considerably reduce the number of times new flows are recorded. For example, using the trace data Abilene and ADSL described in Section 2.5, we find $p = 0.1$ leads to only 15% of flows being recorded in both cases.

On the other hand, the flows that are recorded are the longer-lived flows so that the reduction in the arrival rate of new flows does not translate into a proportionate decrease in the required flow table size. Applying randomized protection with $p = 0.1$ to the Abilene data and the ADSL data with a timeout of 2 seconds, we observe a 70% reduction in the average number of protected flows in the Abilene case, and 63% in the ADSL case.

Note that the gain in occupation increases with the value of TimeOut. Table 1 gives the average and maximum flow list occupation for TimeOut equal to 0, 2 and 20 seconds, together with a measure of the gain G. G is the difference in occupation as a result of applying randomized admission control, divided by the occupation when systematic protection is performed. The results show a significant gain can be obtained.

The gain must be weighed against the impact of the resulting classification errors which can lead to flows being interrupted after having successfully emitted several packets. The resulting ambiguity complicates the realization of applications that must interpret packet discard as flow rejection.

Table 1: Average and maximum occupation for different TimeOut (TO) values, $p = 0.1$

	Abilene			*ADSL*		
	avg	max	G	avg	max	G
TO=0 s	4455	4638	0.37	4244	4434	0.46
TO=2 s	6764	7072	0.70	5072	5300	0.63
TO=20 s	27547	28109	0.83	12499	13027	0.81

3.6 Sizing the tables

It is necessary to size the flow table or the directly addressed structure to limit the probability of saturation to a manageable level. The flow table is saturated when it is impossible to find a free memory location for a newly arriving flow that needs protection. Saturation in the directly addressed structure is less well-defined.

Saturation may be said to occur when the rate of false positive errors exceeds what is judged tolerable. Assuming a maximum false positive rate of p, the structure is saturated when the register occupancy ratio exceeds p. In other words, the latter structure requires $1/p$ times more registers than the number of possible entries in the flow table. However, of course, the registers are typically very small (e.g., only 2 bits in the solution of Sec. 3.4.1) in comparison to the size of the table entries.

The average number of flows in progress may be written FlowArrivalRate $\times$ (FlowDuration + TimeOut) and thus depends on the relative value of the latter two terms. For admission control purposes, a TimeOut value of 2 sec would be sufficient. For the Abilene trace, mean flow duration is 856 ms (many flows have one packet only) and the average size is 5205 bytes.

Extrapolating these characteristics to an OC192 (10Gbps) link loaded to 90% leads to an average of just less than 62000 flows. From simulation results the variance is roughly four times higher. Assuming a Gaussian distribution, the saturation probability is less than 10^{-3} for a table of capacity 64000[1]. The directly addressed structure would require some 640000 registers to meet an assumed tolerable false positive rate of 10%.

The impact of a saturated flow table is that some flows may not be immediately protected. Their packets will be forwarded, however, unless the link is congested and requires admission control. In the directly addressed structure the impact would be to increase the false positive ratio, diminishing the effectiveness of admission control in the event of overload.

Saturation can be controlled for a given protected flow list capacity by dynamically modifying TimeOut. If saturation threatens, a reduction in the value of TimeOut would reduce the average number of flows in progress. The price is the

[1]For an analysis of capacity requirements for a more complex table, see [22]

possible interruption of flows in progress that have an inter-packet interval greater than the new value of TimeOut.

4. CONCLUSION AND PERSPECTIVES

The present work builds on a proposal to enhance the current best effort Internet by making it flow-aware in sharing bandwidth and in controlling accessibility in overload. We have proposed reduced complexity per-flow scheduling and admission control mechanisms and evaluated their performance using trace driven simulation.

We have shown how the complexity of fair queueing, exemplified by Deficit Round Robin, can be reduced by identifying bottlenecked flows in a separate data structure. The size of the DRR schedule can then be limited to around 100 flows, this capacity being attained with low probability (10^{-3}) at relatively high loads (90% utilization). To identify the bottlenecked flows requires an additional low capacity data structure. A memory of just 1000 bits is sufficient for the trace simulations reported in this paper.

To realize the envisaged implicit admission control it is necessary to identify the set of flows that are currently in progress. This can be done using a flow table containing the flow identifier and the epoch of its last packet. The latter could be included in a general purpose flow table used for a variety of applications. Alternatively, in case this is too costly, we have proposed a reduced complexity structure devoted specifically to admission control. In one realization, this structure takes the form of a two-tier bitmap that can be realized simply in hardware. Capacity requirements can be reduced on exploiting the tolerance to imprecise identification of new flows that is inherent to the admission control application.

The flow-aware networking paradigm conforms to the original Internet design philosophy recorded by Clark [5].

- It preserves the *flexibility* for edge-based service creation of the current Internet since the user-network interface is unchanged. Potential for designing new end-to-end applications is increased since fairness no longer needs to be built into the transport protocol.

- Network *survivability* characteristics are enhanced by the possibilities for load sensitive routing brought by admission control. Admission control is not just an alternative quality degradation to reduced throughput but a key element of any adaptive routing scheme.

- *Type of service* differentiation between delay sensitive streaming applications and throughput sensitive elastic data applications is assured without the need for explicit class of service marking. The packet latency of real time flows is particularly small in the fair queueing implementation proposed in this paper.

- While it remains necessary to more thoroughly evaluate the potential for new attacks brought by the flow-aware mechanisms, we believe the greater awareness of network traffic they bring reduces *vulnerability*. Knowledge of traffic at flow level is a frequent requirement for effective intrusion detection schemes.

- Congestion control algorithms, like those of *TCP*, remain essential for users to adjust emissions to the current fair rate. Network assured fairness makes it easier to introduce new transport protocols, better adapted to data transfer at very high speed, for example.

- Last in the list of features identified in [5], flow-aware networking may be considered to improve *cost-effectiveness* and *accountability* by realizing assured performance without the considerable operational complexity of traditional QoS architectures.

We believe, therefore, that flow-aware networking represents a more desirable enhancement to the best effort Internet than current plans for a QoS architecture based on Diffserv-aware traffic engineering using MPLS. We hope the designs presented here will incite vendors to envisage more seriously this alternative and to implement the proposed mechanisms in their routers.

5. REFERENCES

[1] S. Blake, et al., An Architecture for Differentiated Services, RFC 2475, IETF, 1998.

[2] B. Bloom, Space/time tradeoffs in hash coding with allowable errors, Commun. ACM, vol. 13, no. 7, pp. 422-426, July 1970.

[3] B. Braden, et al., Integrated Services in the Internet Architecture: an Overview, RFC 1633, 1994.

[4] Caspian Networks. Flow-Based Routing: Rationale and Benefits. White paper, 2003 http://www.caspiannetworks.com/documents/Apeiro Flow State.pdf .

[5] D. Clark, The design philosophy of the DARPA Internet protocols, Proceedings of Sigcomm '88, August 1988.

[6] A. Demers, S. Keshav, S. Shenker, Analysis and simulation of a fair queueing algorithm, Internetworking: Research and experience, Vol 1, 3-26, 1990. (Also in proceedings of ACM Sigcomm 89).

[7] C. Estan, G. Varghese, New directions in traffic measurement and accounting, Proceedings of ACM Sigcomm 2002.

[8] W. Feller, *Introduction to probability theory and its applications: Vol I*, Wiley International, 1968.

[9] 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.

[10] A. Kortebi, S. Oueslati, J. Roberts, Cross-protect: implicit service differentiation and admission control, Proceedings of HPSR'04, Phoenix, 2004.

[11] A. Kortebi, S. Oueslati, J. Roberts, Implicit service differentiation using deficit round robin, Proceedings of ITC19, Beijing, 2005.

[12] A. Kortebi, L. Muscariello, S. Oueslati, J. Roberts, On the Scalability of Fair Queueing, In Proc. of ACM HotNets III, 2004.

[13] A. Kortebi, L. Muscariello, S. Oueslati, J. Roberts, Evaluating the number of active flows in a scheduler realizing fair statistical bandwidth sharing, In Proc. of ACM Sigmetrics 05, 2005.

[14] A. Kumar, M. Hegde, S. Anand, B. Bindu, D. Thitumurthy and A. Kherani. Non-intrusive TCP connection admission control for bandwidth management of an Internet access link. In IEEE Comm. Mag. Vol 38, No 5, pages 160-167, 2000.

[15] F. Le Faucheur, et al., Requirements for Support of Differentiated Services-aware MPLS Traffic Engineering - RFC 3564, IETF, 2003.

[16] R. Mortier, I. Pratt, C. Clark, and S. Crosby. Implicit Admission Control. In IEEE Journal on Selected Areas in Communications, December 2000.

[17] J. Nagle, On Packet Switches with Infinite Storage, RFC 970, IETF, 1985.

[18] Cisco Systems, Netflow Version 9, 2003

[19] S. Oueslati, J. Roberts, A new direction for quality of service: Flow aware networking, NGI 2005, Rome, April 18-20, 2005.

[20] J. Roberts, Internet Traffic, QoS and Pricing, Proceedings of the IEEE ,Volume: 92 ,Issue: 9 ,Sept. 2004, Pages:1389 - 1399,

[21] M. Shreedar, G. Varghese, Efficient fair queuing using Deficit Round Robin, IEEE/ACM Transactions on Networking, Volume: 4 ,Issue: 3 ,June 1996 Pages:375 385.

[22] J. Xu, M. Singhal, Cost-Effective Flow Table Designs for High-Speed Routers: Architecture and Performance Evaluation. IEEE Transactions on Computers, Vol. 51, No. 9, September 2002.

[23] Y. Zhang, L. Breslau, V. Paxson, S. Shenker, On the characteristics and origins of Internet flow rates, In Proc. of ACM SIGCOMM 2002.