Aggregate Flows – for Efficient Management of Large Flows in the Internet

R. G. Addie, S. Braithwaite, J. das Gupta and J. Leis

Abstract— **Considerable advantages for performance will accrue if large flows can be identified and treated differently in the Internet. DiffServ provides a way to treat flows distinctly in the internet, however configuring DiffServ requires a high level of skill and attention to detail. We seek an approach which is inherently auto-configuring. In this paper flows are not defined uniquely by information in their packet headers, but instead by their route through the local routing domain. This approach imposes a topological structure on the collection of flow by means of which we can judge how *close* two flows are together and, where appropriate, judge two or more flows to be part of a single larger flow. The aim is to get closer to the collection of packets generated by a single human request for service. For example, if a user chooses to subdivide a request as a large number of TCP connections, the entire aggregate would still be identified and treated as a single flow. To reduce the complexity of the task of flow management, we identify only flows larger than a certain threshold, without having to deal with the much larger number of small flows. Simulations are used to determine how many large flows need to be found in order to achieve significant improvements in the performance of the Internet as a whole.**

I. Introduction

There appear to be considerable advantages for performance of networks to be obtained if large flows can be identified and treated differently from the remaining Internet traffic [8]. Methods for identifying flows in routers have been an active topic of research for some time [5], [10], [12]. In this paper a new definition of flow – attempting to match a single user service request – is adopted and it is shown that it is possible to identify the *large flows*, flows larger (in total bytes) than a certain threshhold, without having to deal with the much larger number of small flows. Since the total number of flows is very large whereas the number of large flows is quite small, being able to identify the large flows without having to identify the remaining flows provides considerable advantages and may well be the critical step in ensuring scalability of a performance management system based on flows.

The definition of a flow is a matter for careful consideration. We take the view in this paper that the key features of a flow should be: (i) its path through the local routing domain – all packets in a flow follow the same path in the local domain; and (ii) statistical correlation. By identifying flows by means of their local path together with statistical characteristics, rather than by specific details of protocol use, the large underlying user-generated flows can be identified despite arbitrary choices of how these flows are implemented. For example, if a user chooses to subdivide a request as a large number of TCP connections, the entire aggregate would still be identified and treated as a single flow according to our definition.

Since the objective of identifying and treating large flows differently is to improve overall network performance it is clear that the behaviour of these flows (the fact that they are large) is sufficient to identify them. Any *indirect* method of identifying flows, i.e. a method not based on their behaviour, will only reduce the accuracy with which these flows can be targeted. We therefore adopt this approach as a matter of principle.

In [5], [10] and [12], for example, it is assumed that the identity of the flow containing a packet can be unambiguously deduced from information in each packet. For this reason, the traditional methods for identifying large flows are not applicable to the definition of flow adopted here. The complexity of the flow identification algorithm proposed here is a modest amount of per packet processing in conjunction with $O(N)$ fast memory locations, where N is the maximum number of large flows which need to be monitored. This is similar to the algorithms discussed in [5] and [10] except that the number of large flows needing separate attention, and therefore N, is smaller in the present case. This will be discussed in more detail in Subsection IV-B.

The key performance criterion for the network that we concentrate on in this paper is the *relative response time* (RRT) delivered to flows. The RRT of a flow is the time between the arrival of the flow and its complete delivery to its destination, divided by the duration which would be necessary in an unloaded network. We argue that statistically bounded RRT, e.g. RRT < 5 with probability 99 %, is a satisfactory form of guaranteed quality of service (QoS). This form of guaranteed QoS is theoretically convenient because a single measure of performance is used for all services. This guarantee can be provided for small flows by giving them priority over large flows and for large flows the problem is much easier because they operate over a longer time scale and therefore the relative response time criterion is much easier to meet. Low to modest rate CBR services, such as VoIP, should be regarded as a series of short flows, not as one large flow. It is important that the flow identification algorithm does not mistake a long CBR communication as a large flow. The proposed algorithm has this property for CBR flows below a certain rate. A CBR communication of a higher rate does risk being treated as a large flow, which is to be expected, since such a flow does have the potential to disrupt other flows.

It has been shown in a related work in preparation at the same time as this paper that if traffic takes the form described in Section III, statistically guaranteed RRT (relative response time) can be provided by adequately dimensioning links so long as either the *shortest job first* (SJF) or *Fair Queueing* (FQ) queue disciplines are used across flows in all routers. We shall see in Section V that the SJF queue discipline appears to deliver significantly better RRT performance than FQ. The SJF discipline is impractical, however there is a range of queue disciplines intermediate between SJF and FQ which are implementable and approach the performance of SJF. The discpline x-SJF/FQ treats flows longer than x (in bytes) according to the SJF discipline, and the remaining flows according to FQ. Flows shorter than x are given priority over flows longer than x. We shall see in Section V that for a reasonably large value of x, the x-SJF/FQ queue discpline delivers performance similar to SJF.

In Section II, the overall framework by means of which this work contributes to an architecture for guaranteeing quality of service in the Internet is described. In Section III, a traffic model, similar to the widely used $M/G/\infty$ traffic model discussed in [11], [13], is described and key features discussed. In Section IV, the algorithm for identifying large flows is described and its complexity analysed. In Section V, simulations which demonstrate the appropriate size of flow to single out and how this size of flow may be identified is investigated. Difficulties and issues in regard to implementing the proposed queueing algorithm, and a specific implementation which has already been completed, are discussed in Section VI. Finally, conclusions are drawn in Section VII.

II. Network Context

The motivation for this paper is to formulate and justify an architecture for guaranteed quality of service. The quality of service philosophy under consideration is characterised as follows:

(i) quality of service in the Internet is compromised, at present, by unpredictable performance in access networks (core networks can deliver statistically guaranteed performance already);

(ii) performance in the Internet can be statistically guaranteed by ensuring that RRTs have a well-defined probability distribution with variance tending to zero as utilization reduces;

(iii) the FQ and SJF queueing disciplines can ensure that RRT has a well-defined probability distribution with variance tending to zero with utilization – as required in the previous condition;

(iv) the SJF queueing discipline is significantly better than the FQ discipline in achieving low relative response time.

(v) there is a definition of *flow* in the Internet which is adequate for guaranteeing flow-level performance and in which flows can be identified sufficiently quickly in routers that flow-based AQM's can be practically and scalably implemented, and provide an adequate approximation to the SJF discipline.

Research already published supports some of the above, with the notable exceptions of (v). In particular, (i) is strongly supported by [4] and [6].

Hypothesis (ii) is highly plausible and a simple proof can be given directly if we assume the standard deviation forms a scale parameter of the distribution of RRT. Suppose we wish to bound network

performance to be less than ε with probability δ. This requirement can be achieved, by chosing the standard deviation which meets this standard and then choosing the maximum utilization consistent with this setting for standard deviation.

Hypothesis (iii) is addressed in [1]. Hypothesis (v) is the subject of this paper. Hypothesis (iv) remains for future work although strong evidence for it is also provided in Section V.

It follows from these six hypotheses that quality of service can be statistically guaranteed, in the Internet, by setting utilization at appropriate levels in each link and using either FQ or SJF to decide how to treat flows at each router in access networks. It should be noted that although SJF probably cannot be scalably implemented, there is a compromise queue discipline, x-SJF/FQ, which is SJF for the large ($> x$) flows and FQ for the remaining, which can be implemented scalably and which, we shall see in Section V achieves most of the benefits of SJF.

The role of this paper is mainly to demonstrate hypothesis (v), that there is a definition of *flow* which is *closer* to matching user generated requests and which allows *large flows* to be efficiently identified and separated from other flows. The definition of this paper is a step in the right direction to matching user-generated flows more closely in the sense that users' flows do not neatly map onto atomic TCP connections, but rather to collections of TCP connections,

The way this more user-friendly definition of flow is effected is to successively refine the traffic passing through the router into categories, sub-categories, sub-sub-categories, and so on, in such a way that at each stage only a moderate proportion of the categories under consideration need further attention.

There may be a perception that this hypothesis has been well addressed in existing work, e.g. [5], [10], [12], including existing software in routers. However, there are two strong reasons for reserving judgement in this regard and therefore tackling this issue with considerable vigour: (i) work undertaken in parallel with this paper [1] now strongly suggests that the benefits of accurately identifying and controlling large flows is considerable; (ii) if we are to rely on the identification of large flows to a considerable degree in router queue management, it is essential to have the right definition.

III. The Traffic Model

The traffic model used here is that all packets in the Internet are contained in flows, the *arrival times* of which form a Poisson process. Flow lengths, e.g. X, have the Pareto distribution

$$P(X > t) = \begin{cases} \left(\frac{t}{\delta}\right)^{-\gamma}, & t > \delta, \\ 1, & \text{otherwise.} \end{cases}$$

We do not expect this model to be literally true, but the important features of real traffic are present. In particular, the true distribution of flow lengths is probably not exactly the Pareto distribution, but the important feature of heavy tails is present in real traffic and is well modelled by the Pareto distribution. Also, the correlation in byte flows in real networks introduced by the flow structure of this model does appear to explain a considerable amount of the correlation in real networks [9].

IV. The Algorithm

The main part of the algorithm is continuous measurement, by means of token buckets, of certain aggregates of flows, identified by the path along which they are routed. Packets are routed to an appropriate token bucket in exactly the same way that they are routed to the next hop along their path, and so we will call it a routing table. The entries in this token bucket routing table change dynamically, however, depending upon the current conditions. The procedure for updating this token bucket routing table is the complex part of the algorithm, although the computational load of this aspect should not be too large since it only applies when an aggregate flow makes a transition from being classified as not containing a large flow to containing one, or conversely.

The algorithm deals always with *aggregates* of flows. Individual flows might be obtained, eventually, by this algorithm, if necessary, by a process of successive subdivision of aggregates into smaller and smaller aggregates. The subdivision is on the basis of the path followed. A large aggregate is defined as the packets following a certain short path during their traversal of the Internet. A subaggregate is formed by lengthening this common path. This action of lengthening or shortening the path corresponding to an aggregate flow is represented in Figure 1 by the node labelled "Update RT".

At each step in this process of refinement, we need to identify whether the aggregate contains a large flow, as defined by a threshhold. This measurement is made by a token bucket tailored for the purpose.

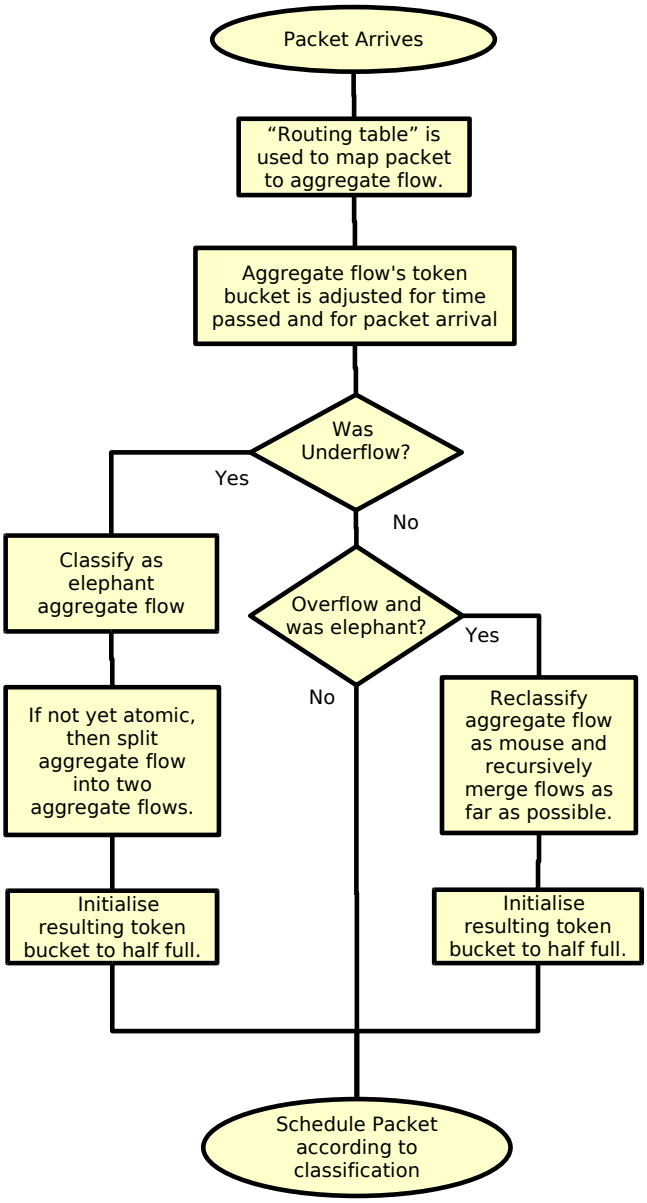

Fig. 1. The per-packet algorithm

A simplified flow chart of the per-packet aspect of the algorithm is shown in Figure 1. This is expressed below in more detail:-

1) Data Structure:

- There is a "routing table" with masks and values to match. A mask of length N bits will have an associated value of N significant bits.
- One may be able to split an aggregate flow with a mask of length N into two aggregate flows of length N+1. The two resulting values will then differ only in the (N+1)th bit.
- There is a point beyond where an aggregate flow is said to be atomic and cannot be further split.
- It is possible to merge two entries of length N+1 that have values that differ only in the last bit. The resulting entry has length N.
- Each entry in the routing table is associated with a record for an aggregate flow.
- Each aggregate flow has a token bucket associated with it.
- Each aggregate flow can be classified as mouse or as elephant.
- Packets that are being processed can be classified as mouse or as elephant.

2) Initialization: Initially there is only one aggregate flow encompassing the IP domain of the router, and this aggregate flow has the classification of mouse.

3) For each packet arrival:

- Check the source (or destination) IP address against our routing table and find the matching aggregate flow record. This is done by bitwise ANDing the IP address with the mask and then comparing the result with the value. The first match is always accepted.
- Request the token (or tokens) from the token bucket and add the appropriate amount of tokens for the time that has passed since this token bucket was last touched.
- IF this is an aggregate flow with the classification of elephant THEN
  - Set the classification of the packet to elephant.
  - IF token bucket has overflowed THEN
    - Set the classification of the aggregate flow as mouse.
    - Set the classification of the packet to mouse.
    - Recursively aggregate mouse flows as far as possible.
    - Initialize the resulting token bucket to half full.
  - ELSE IF the token bucket has underflowed AND the flow aggregate can be split THEN

    - Split the aggregate flow into two aggregate flows.
    - Initialize the resulting token buckets to half full.
- ELSE
  - Set the classification of the packet to mouse.
  - IF the token bucket has underflowed THEN
    - Set the classification of the aggregate flow as an elephant.
    - Set the classification of the packet to elephant.
    - IF the flow aggregate can be split THEN
      - Split the aggregate flow into two aggregate flows.
      - Initialize the resulting token buckets to half full.
    - ELSE
      - Initialize the token bucket to half full.

A. Token Bucket Parameters

Identifying whether a flow aggregate contains a large flow will be achieved by means of a token bucket . Selection of the parameters of these token buckets needs to be robust and stable, i.e. small errors in the choice of parameters will not critically affect the outcome. A typical situation is depicted in Figure 2. When the node is located in the middle of its routing domain a network which includes a tree on both sides of the node would be necessary, however this additional complexity can be neglected to simplify the discussion here without compromising the argument. The only large flow aggregate in this case is the aggregate of flows on the uppermost path of the network shown. The token bucket parameters must be selected to identify when a large flow becomes small, and conversely.

In order to set the token bucket parameters we need to know what typical *rates of arrival of bytes* should be for each aggregate flow, both when it contains a large flow, and when it does not. In the former case, the rate depends upon how many large flows are active for this outgoing path.

A fundamental assumption of the traffic model is that long term fluctuations of traffic level are mainly due to large flows. Traffic due to short flows should not cause an overload of an outgoing link. Denote

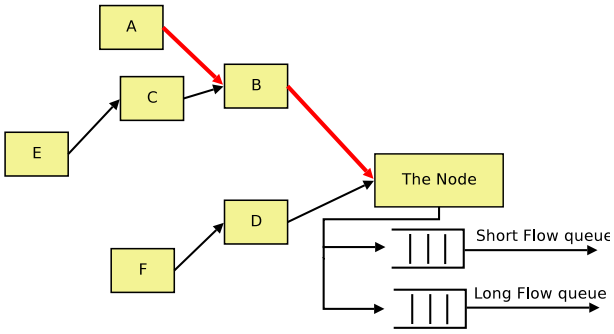

Fig. 2. Typical configuration of large and short flow routing

the capacity normally required by the short flow traffic by C_S and the total capacity on the link by C. There should therefore be a significant capacity, $C_L = C - C_S$ say, which is effectively reserved for the large flows. When an undetected large flow is present, a significant proportion of C_L will be used by this large flow. This additional traffic will show up in the bytes enqueued as short flow traffic and in the token bucket for the aggregate flow which has changed state from short to large. These events should trigger an update of the token bucket routing table, as indicated in Figure 1.

In the case when a large flow becomes small, the traffic flow on the path in question will, likewise, drop from $\alpha C_S + \beta C_L$ to αC_S, where α is the expected proportion of short flow traffic on this path, and β is the proportion of long flow traffic on this path. If there are currently N long flows, an appropriate value of β is $\frac{1}{N}$.

B. Complexity

The algorithm of this paper is not strictly comparable to most existing algorithms, since it deals with aggregates of flows rather than with specific flows. The algorithm of [10] has processing time of order $O(1)$, per packet, and also makes use of N memory locations, where N is the desired number of large flows. The first algorithm of [5] similarly has time complexity $O(1)$ per packet and space complexity $O(1)$ per large flow.

The per packet processing of the present algorithm is again $O(1)$ per packet and in fact it is necessary to take an action in connection with every packet. The number of memory locations referenced by the algorithm is proportional to the number of *aggregates* of flows being monitored at any one time. Since the number of aggregates of flows is

expected to be $O(1)$ times the desired number of large flows, the complexity should be superficially similar to the other algorithms.

As discussed in [5], the space complexity of these algorithms is critical to their effectiveness because if a large quantity of memory is required this memory will need to be of lower cost and therefore lower speed, which will reduce processing capacity. Dealing with aggregate flows is an advantage because there are many fewer aggregate flows than end-to-end flows. In fact we need to consider the space complexity of the algorithm relative to the total network size, measured by total connected hosts.

Denote total network size by H, end-to-end flows by N_e and aggregate flows by N_a. For the moment, let us assume that the threshhold above which a flow is declared to be large is fixed. If routing domains increase in size at all as the total network increases, it is certainly at an order below 1, hence $N_a \sim O(H^\nu)$ for $\nu \ll 1$. On the other hand, as the total network increases in size, we expect each host to be in communication with more and more hosts and each router to have to deal with more flows, hence $N_e \sim O(H^\mu)$ for $1 \geq \mu \gg \nu$

As the number of flows increases the *proportion* we need to deal with as large will not stay the same, but will in fact reduce. This is because the choice is governed by the criterion that we reduce the standard deviation of the short flow traffic to less than $C_L/3$. As the network increases, the standard deviation of the short flow traffic increases in proportion to $\sqrt{H}$. This makes it significantly easier to monitor and control large flows, or large aggregate flows. Furthermore simulations indicate that the number of aggregate flows which need to be monitored is quite modest and readily achievable.

In summary, we expect space complexity of the algorithm of this paper to be $N_a \sim O(H^{\nu/2})$ whereas algorithms which deal with end-to-end flows will have space complexity $N_e \sim O(H^{\mu/2})$, in which $0 < \nu \ll \mu < 1$. Moreover, experiments in the next section indicate that $N_a \ll 100$ in a typical router, ensuring that the proposed algorithm should not place undue processing strain on the router.

V. Simulation Experiments

A number of simulation experiments, using a Poisson aggregate of Pareto distributed flow lengths, have been conducted, with a view to identifying an appropriate threshhold for the definition of a large flow and how many such flows will need to be monitored. The Poisson-Pareto model exhibits the statistical features of Internet traffic which are relevant to this particular issue. Results from some of these simulations are shown in Figure 3. This particular figure applies to flows of length 1-2. Note that the SJF and s-SJF/FQ curves are indistinguishable in this graph.

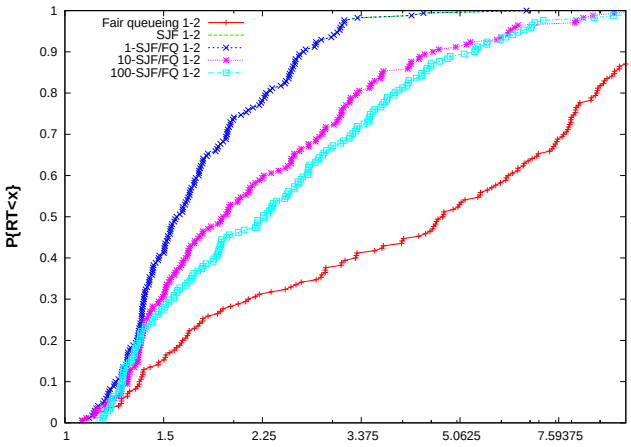

Fig. 3. Comparison of SJF and FQ

This figure shows the RRT for similar experiments in which various queue discplines at a router are simulated. The queue disciplines include SJF, FQ, 1-SJF/FQ, 10-SJF/FQ and 100-SJF/FQ. The other parameters of these simulations are: $\gamma = 1.1$, $\delta = 0.01$ (the length of the smallest flow), $\lambda = 10$ (the arrival rate of new flows is 10 per second). The occupancy (proportion of busy time) of this system is approximately 0.75.

The message from these experiments is that:

(a) There is a significant difference in the RRT which can be delivered by a router which implements SJF, relative to one which implemenets FQ. This is clear from Figure 3. The figure shows the estimated probability distribution of RRT provided, for flows of lengths 1-2, under these two and other queue disciplines, as recorded in a simulation over 20,000 seconds.

(b) Most of advantage of SJF are achieved by x-SJF/FQ, which only finds and treats flows longer than x according to the SJF discipline, for values of x sufficiently large to ensure that the queue discipline can be practically implemented.

The number of long bursts which need to be monitored turns out to be very low. For example, when the 100-SJF/FQ discipline is used, the average

number of long bursts at any time is 0.85 over a simulation of length 200,000 seconds. Also, the rate at which changes of state occur is also very low. A plot of the number of flows of length 100 or more in a simulation of 20,000 seconds is shown in Figure 4.

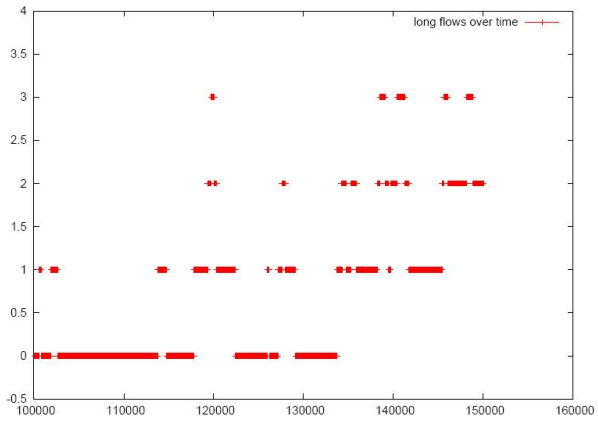

Fig. 4. The number of long flows over time

VI. Implementation

An implementation of an algorithm similar to the one described above in broad features has been undertaken [3]. This work demonstrates the feasibility of implementing the proposed algorithm. This was an implementation of a Mice and Elephants queuing discipline, where mouse packets were given absolute queuing priority over elephant packets. The overall dropping probability was determined by ARED [7], and mouse packets were given immunity to dropping.

The implemented algorithm keeps records of each active flow according to the properties in the packet headers. The proposed algorithm differs from the implemented algorithm in that it will benefit the mouse flows without requiring a large amount of computer memory in order to track individual IP flows. To do this, the proposed algorithm needs to keep a token bucket routing table. The implemented algorithm uses bytes since the beginning of a flow as a measure of flow length. The proposed algorithm uses a token buckets in order to classify flows as small or large.

The development was carried out on Linux because of the ready availability of the source code. In Linux, queueing disciplines run in the Linux kernel itself, which can be a daunting place for code development. Code in the Linux kernel enjoys no protection from errors, and it is not convenient to retrieve debugging information from there.

The development was achieved by means of a queuing discipline development environment [2] which duplicated Linux's queueing discipline interfaces so that queuing discipline code destined for the Linux kernel could be developed in user space. Thus, the development of the code was rendered much easier than it otherwise could have been.

VII. Conclusion

This paper has investigated some critical issues in relation to the concept of providing a guarantee of QoS by ensuring that long flows are treated differently from short ones in routers. This concept is very attractive because it potentially establishes a method whereby a guarantee of QoS can be provided without a new Internet network architecture, and without tagging packets with class of service labels. The proposed method relies on the fact that the benefits of treating longer flows with lower priority than shorter ones are so great, for all flow lengths, that assigning quality of service on the basis of *need* becomes unnecessary. However, for this approach to work we probably need a different concept of flow than has been used in the literature up to now. It is important that the flow concept adopted errs by overaggregating flows together rather than by potentially make overfine distinctions. The errors from overaggregation can be addressed by other mechanisms. However, a definition of flow which is too technical, and which therefore subdivides user requests artificially, will potentially be ineffective in the critical goal of protecting short user requests from the degradation caused by other flows.

The simulations undertaken in this work, results of some of which are presented here, demonstrate the number of flows which need to be monitored is quite small and that an efficient and scalable queue discipline exists which can undertake the necessary monitoring and control.

References

[1] R. G. Addie and M. Zukerman. Use of the dense poisson pareto process to demonstrate guaranteed flow performance. Technical report, University of Southern Queensland, 2006.

[2] Stephen Braithwaite. Creating new linux queuing disciplines howto. June 2005. URL: `http://www.sci.usq.edu.au/staff/braithwa/MastProj/howto.html`.

[3] Stephen Braithwaite. Implementation of aqms on linux made easy. *accepted for presentation at the Open Source Development Conference*, 2006.

[4] Jin Cao and Kavita Ramanan. A Poisson limit for buffer overflow probabilities. In *Proceedings of IEEE INFOCOM 2002*, volume 2, pages 994–1003, 2002.

[5] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3):270–313, August 2003.

[6] Do Young Eun and Ness B. Shroff. The impact of aggregation in simplifying network analysis. In *Proceedings of INFOCOM 2002*, New York, NY, Jun 2002.

[7] S. Floyd, R. Gummadi, , and S. Shenker. Adaptive red, an algorithm for increasing the robustness of red's active queue management. *ACM SIGMETRICS Performance Evaluation Review*, 3, June 2001. URL: `http://www.icir.org/floyd/papers/adaptiveRed.pdf`.

[8] Liang Guo and Ibrahim Matta. The war between mice and elephants. Technical Report 2001-005, Computer Science Department, Boston University, MA 02215, 7 2001. citeseer.nj.nec.com/guo01war.html.

[9] Nicolas Hohn, Darryl Veitch, and P. Abry. Cluster processes, a natural language for network traffic. *IEEE Transactions on Signal Processing, Special Issue on Signal Processing in Networking*, 51(8):2222–2249, 2003.

[10] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems*, 28(1):51–55, March 2003.

[11] N. Likhanov, B. Tsybakov, and N. D. Georganas. Analysis of an ATM buffer with self-similar ("fractal") input traffic. In *Proceedings of IEEE INFOCOM 1995*, pages 1–15, April 1995.

[12] K. Chen M. Charikar and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, pages 3–15, 2004.

[13] M. Mandjes. A note on queues with M/G/∞ input. *Operations Research Letters*, 28(5):233–242, 2001.