Local Flow Separation

Vincenzo Liberatore
Division of Computer Science
Case Western Reserve University
10900 Euclid Avenue
Cleveland, Ohio 44106-7071, USA
vincenzo.liberatore@case.edu
http://home.cwru.edu/~vxl11/

***Abstract*— This paper elaborates on a paradigm for Quality-of-Service (QoS) that is *local*, i.e., it does not depend on multi-node cooperation. In order to maintain short queuing delays, we individuate flows that occupy a large fraction of a buffer and segregate those flows into a separate queue. The algorithm is provably fair and can avoid all packet re-orderings. We show through extensive simulations that state requirements are minimal, and that other flows will benefit from short queuing delays while aggressive flows can still maintain high throughput.**

I. Introduction

The deployment of Quality-of-Service (QoS) has reached an impasse that is largely caused by the very design philosophy of the Internet. Decentralized control and autonomy mandate an open architecture with no global control. A decentralized architecture is partly responsible for preventing the adoption of strategies that rely on multibox administration or inter-provider coordination. Another fundamental principle is best-effort [1], which implies unreliable service with no QoS provisioning. A best-effort network can still guarantee statistically adequate QoS if over-provisioned. On the other hand, best-effort networks do not limit or separate flows and so these networks can in principle give rise to poor levels of service. Figure 1 shows a straightforward but concrete example of the impact of a single bulk data transfer on Round-Trip Times (RTTs). In general, over-provisioned best-effort networks can work well in the typical case, but introduce an element of risk that cross-traffic can congest links and cause poor QoS. On the whole, QoS should take into account fundamental architectural principles (best-effort, decentralized control) and at the same time overcome the inherent congestion risks of the best-effort model.

A general paradigm for QoS in best-effort networks is based on *local mechanisms to protect from global risk*. In this paper, we take one step in this direction by investigating a method to maintain short queues in best-effort networks that are generally over-provisioned but occasionally congested and present little support for distributed coordination. The YAQS (Yet Another Queuing Strategy) algorithm attempts to obtain short queuing delays in a best-effort router that is crossed by aggressive flows. The main idea involves the following two elements. First, a hybrid method maintains per-flow queue occupancy with high accuracy and limited state. Second, when the queue occupancy of a certain flow exceeds a threshold value, further packets belonging to that flow are segregated in a different queue. The forwarding decision is then made by an approximately fair schedule between the regular and the segregated queue. The intuition is that flows with small buffer occupancy should benefit from a shorter queue once the more aggressive flows have been separated. Main issues include queuing delays and classification precision. Secondary issues are packet reordering and flow goodput.

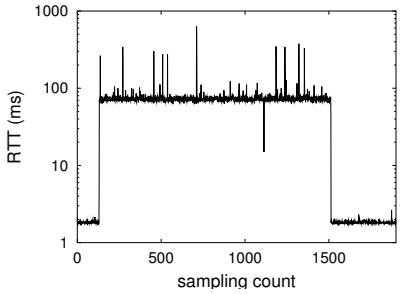

Fig. 1. Ping (ICMP) RTTs collected between two machines across campus. During sampling period 131 through 1512, a *single* TCP flow is active between two other machines whose end-to-end path shares a link with the ping flow.

The next section gives the relevant background on these issues. Section III gives a high-level outline of our queue management algorithm. Section IV discusses the evaluation methodology. Section V describes the impact of such algorithm on queue length. A summary of related work, conclusions, and future work end the paper.

II. Background

Aggressive Flows: An aggressive flow is a flow that congests a link and consequently causes queuing delays and packet losses. Aggressive flows arise for a variety of reasons. The most benign type of an aggressive flow is a TCP bulk data transfer (e.g., Figure 1), especially if it has a small RTT or large end-point buffers. Such a flow backs off if it detects congestion, but it also probes constantly for additional bandwidth and, as a result, can saturate the buffer on a bottleneck link. The benign type of aggressive flow is common especially in access networks due to their bandwidth limitations [2], but it also present in the core [3]. More malicious flows

include unregulated flows or selfish behaviors. All aggressive flows lead to poor QoS, regardless of whether they were generated by well-behaved (e.g., TCP) or ill-behaved (e.g., CBR) end-points. Finally, a flow aggregate is a set of flows that originate from a single or from multiple clients and that jointly exhaust network resources. The best solution for aggregates are to improve protocols (e.g., persistent connections [4], multicast [5], persistent dropping [6], peer-to-peer networks) or to explicitly monitor traffic and take explicit remedial actions [7], [8]. In this paper, we will not explicitly address aggregates and assume instead that aggregates are dealt with by a specialized upstream detection systems (e.g., Figure 1 in [7]) or by protocol development.

Queuing: Short queues can be guaranteed by limiting the router buffer size, but at the cost of packet losses with bursty traffic [9] and consequent throughput reduction. Another obvious approach would be to classify flows as aggressive and drop all packets from those flows, but at the cost of unfairly penalizing regular bulk data transfers. A better solution is to explicitly provide for flow separation through Fair Queuing (FQ) or one of its variants. However, FQ's more scalable versions are either non-deterministic and can require a large number of buckets [10] or necessitate explicit and coordinated packet classification [11].

Active Queue Management (AQM): An alternative to FQ is to employ AQM, such as Random Early Drop (RED) [12], to curb the sending rate of aggressive flows. AQM is relevant in best-effort networks if no explicit end-system cooperation is required. For example, AQM could employ packet dropping instead of or in addition to marking. Even so, aggressive flows would be punished but there is no guarantee that the end-points would necessarily moderate their sending rates and alleviate congestion. Although various AQM strategies can lead to high link utilization, throughput fairness, and short queues on average, AQM does not protect against periods of long queuing delays ([13], see also Figure 8(c) for an example). The crux of the problem is that AQM dynamics depend on end-to-end RTTs and can consequently have a relatively slow adaptiveness, which in turn can lead to long queues. In particular, even if an AQM methods can set a small reference queue length, the convergence to that short queue can be slowed down by the dependency on RTTs. The most obvious dependency of AQM dynamics on RTTs arises from the propagation of congestion indicators to the source, and this communication can require a significant fraction of the end-to-end RTT. A more subtle dependency on RTTs stems from the need to maintain stability in the control-theoretical sense [14], [15], [16], [17]. Stability typically requires that indications of congestion be weaker when the flow RTT is large. For example, a drop (or marking) probability would increase more slowly if flows have longer RTTs. In turn, a weaker congestion signal (e.g., a slowly changing drop probability) delays the AQM dynamics and can lead to extended periods of long queuing delays. The dependency on RTTs is shared among AQM strategies and is actually the manifestation of the more general control-theoretical trade-off between stability and tracking in the presence of delays in the feedback loop (e.g., [18], [14]). AQM's sensitivity to RTTs motivates the investigation of strategies that can adapt quickly and, in particular, that are local to each router. An example of such strategies is FQ, with the potential problems described above.

Detection: At one extreme, exact queue occupancy can be maintained in a hash table (e.g., [19]). At the other extreme, approximate occupancy can be derived from a small random sample of the packets in the queue (e.g., [20]). Hash tables are accurate but require per-flow state and, conversely, random sampling is approximate but does not require per-flow state. An intermediate solution is to maintain approximate queue counts (e.g., [21], [22], [23], [24]).

III. YAQS

A. Algorithm

The YAQS queuing strategy is shown below as Algorithm 1 and 2. Upon arrival of a packet p, YAQS checks whether the corresponding flow occupies $\bar{c}_f$ bytes in the fast queue or $\bar{c}_s$ bytes in the slow queue. If so, the packet is enqued in the slow queue, otherwise it is enqued in the fast queue (Lines 1–3). The slow queue occupancy $\tilde{c}_s$ is an exact value whereas the fast queue occupancy $\tilde{c}_f$ is an estimate obtained with an approximate method. The motivation for using two counters is their ability to maintain accurate queue occupancy estimates with minimal state, as discussed in more details in Section V. Furthermore, a broader consideration is that an administrator could benefit from an exact knowledge of which specific flows are occupying a large fraction of a buffer. After an appropriate queue has been chosen, the packet is enqued (or it is dropped if there is no more buffer space, Lines 4–7). If the packet is assigned to the slow queue, its logical length can be inflated (Lines 9–15) to avoid a certain type of reordering, as discussed in Section III-C. The remaining details of Algorithm 1 will be introduced gradually in the rest of the paper.

The deque algorithm (Algorithm 2) is an approximation of an ideal Generalized Processor Sharing (GPS) between the fast and the slow queue [25]. The GPS algorithm creates two logical links of bandwidth proportional to $B/(1+r)$ for the fast queue and $rB/(1+r)$ for the slow queue, where B is the raw link bandwidth and r is a tunable parameter. However, GPS is also work-conserving: if either queue is empty, the other queue can forward packets. GPS can be approximated with various methods, and Algorithm 2 is a simple method especially tailored for the two YAQS queues. The general objectives of this deque algorithm are simplicity, fairness especially with respect to the fast queue (Section III-B), and the avoidance of re-ordering in conjunction with packet length inflation (Section III-C). The basic idea of the algorithm is to credit the slow queue for each byte that is forwarded from the fast queue and to schedule the slow queue only when its credit exceeds the inflated length of its head-of-line.

It is useful to briefly compare YAQS with the other strategies mentioned in Section II. In the first place, YAQS supports large buffer but simultaneously aims at a short fast queue.

Algorithm 1 YAQS: Enque

Require: p is an incoming packet, $\bar{c}_f$ and $\bar{c}_s$ are queue occupancy thresholds for the fast and the slow queue respectively, r is the weight of the slow queue normalized to the fast queue weight ($0 \leq r \leq 1$), credit is a variable maintained by the deque algorithm (Algorithm 2)

Ensure: p is assigned to either the fast or the slow queue

1: $\tilde{c}_f \leftarrow$ (estimate) fast queue occupancy of the flow to which p belongs
2: $\tilde{c}_s \leftarrow$ (exact) slow queue occupancy of the flow to which p belongs
3: $q \leftarrow \tilde{c}_f > \bar{c}_f$ or $\tilde{c}_s > \bar{c}_s$? slow queue : fast queue
4: **if** length of p exceeds buffer remaining in q **then**
5: Drop p
6: **else**
7: Enqueue p in q
8: Add the size of p to the per-flow occupancy counters of q
9: **if** q is the slow queue **then** {Packet length inflation}
10: $Q_f \leftarrow$ length of the fast queue
11: $Q_s \leftarrow$ length of the slow queue (including p and all nop bytes)
12: **if** $rQ_f - Q_s + \text{credit} \geq 0$ **then**
13: Inflate the length of p by $\lfloor rQ_f \rfloor - Q_s + \text{credit} + 1$
14: **end if**
15: **end if**
16: **end if**

Algorithm 2 YAQS: Deque

Require: r is the weight of the slow queue normalized to the fast queue weight ($0 \leq r \leq 1$), credit is initially set to 0

Ensure: q is the queue whose head-of-line is transmitted

1: **if** either queue is empty **then**
2: choose a non-empty queue q if possible
3: credit $\leftarrow 0$
4: **else if** credit $\geq$ inflated length of slow head-of-line **then**
5: $q \leftarrow$ slow queue
6: credit $\leftarrow$ credit $-$ inflated length of slow head-of-line
7: **else**
8: $q \leftarrow$ fast queue
9: credit $\leftarrow$ credit $+r\cdot$ (length of fast head-of-line)
10: **end if**

YAQS does not require packet classification other than locally at the router and, unlike FQ, it involves only two queues. YAQS is more direct than AQM: no end-to-end RTTs are involved in the algorithm dynamics since all decisions are made locally. YAQS is not preclusive of AQM strategies, which can be employed for example to manage the slow queue. Incidentally, this paper focuses on queue lengths, and we will investigate non-real-time throughput only inasmuch as no direct negative consequence arises from the packet separation into the slow and fast queues. Fairness among all (slow) flows is beyond the scope of the paper. By the same token, the paper focuses on queuing delays, and other QoS metrics, such as bandwidth, will not be considered directly here.

A remark regards the distinction between the slow and the fast queue. If $r = 1$, the forwarding decision is a balanced round-robin between the two queues, and so the slow queue is in some sense as fast as the fast queue. However, there is no explicit attempt to reduce queuing delays in the slow queue, and hence its name of "slow".

B. Fairness

A property of YAQS is that it approximates GPS, as per the following analysis. We make the distinction between the time instant when a packet is deque (i.e., it has been scheduled for forwarding and its transmission begins) and the time instant when a packet departs (i.e., its transmission completes). The departure and deque time differ exactly by the packet transmission time.

Definition 1 ([26]): A deque algorithm has *time worst-case index (T-WCI) equal to P for the queue q* if a packet that arrives at time t in the queue q will depart at or before time $t + P + Q/B_q$, where Q is the length of q at time t (including the packet that arrives at time t and all "nop" bytes if any) and B_q is the bandwidth assigned to q by GPS.

The proof of the following proposition is omitted for compactness.

Proposition 1: Let P_s be the maximum transmission time of a packet enqueued in the slow queue and P_f be the maximum transmission time of a packet enqueued in the fast queue. Algorithm 2 has time worst-case index (T-WCI) equal to $P_s + rP_f$ for the fast queue and equal to $2P_f$ for the slow queue.

A large value of r increases the fast T-WCI. Thus, $0 \leq r \leq 1$ is assumed in Algorithm 2. In addition to its fairness properties, YAQS can also avoid packet re-orderings, as discussed next.

C. Re-Orderings

The presence of two queues can in general create packet re-orderings, which can negatively impact end-to-end performance. For example, in the case of a TCP flow, packet reordering has effects ranging from an increase in the number of acknowledgment packets to the degradation of flow goodput. Re-orderings fall in two categories: a *fast-after-slow* reordering is a packet that is enqued in the fast queue but is forwarded after packets that belong to the same flow, that arrived after p, and that were enqued in the slow queue. Symmetrically, *slow-after-fast* re-orderings are packets enqued in the slow queue that are forwarded after subsequent fast packets from the same flow.

Proposition 2: Algorithms 1 and 2 avoid all fast-after-slow re-orderings.

Proof: (Proof Sketch.) The property follows from packet length inflation (Lines 9–15 in Algorithm 1). The rest of the proof is omitted for lack of space. ■

Although packet length inflation removes fast-after-slow reorderings, it also effectively increases the amount of traffic

Id	Packets	BW
1	1335082	131.3
2	385075	23.9
3	1344755	93.9
4	1573240	80.8
5	2285605	123.4
6	330444	11.9
7	818536	44.7
8	646023	25.6

TABLE I
CHARACTERISTICS OF THE NLANR PACKET TRACES.

that a slow flow must transfer. Thus, it remains to verify that packet length inflation introduces a relatively small amount of pseudo-traffic (Section V). As for slow-after-fast re-orderings, it is easy to see that they are avoided if $\bar{c}_s = 0$. This parameter choice $\bar{c}_s = 0$ will be further discussed in Section V.

IV. EXPERIMENTAL METHODOLOGY

Two types of experimental methods have been considered. The first involves NLANR *packet traces*. Table I shows certain characteristics of our packet traces. These packet traces were collected in the early afternoon of a 2003 mid-week day on OC-3 links and represent roughly 90 seconds of traffic. BW is the average link bandwidth utilization expressed in Mbps. Most traces log packets at the time when they are forwarded on a link, and so they do not present a queuing behavior. However, queuing is critical for our analysis, and so we assume that the links where the trace was collected directly feed into another link with smaller bandwidth. Since the downstream bandwidth is smaller than that of the link where the trace was collected, queues can form. The downstream buffer is assumed to have infinite capacity.

Definition 2: The *over-provisioning factor* β of a link during a certain time interval T is the ratio of the link bandwidth over its average utilization during T.

In the rest of the paper, the downstream bandwidth will be expressed as its over-provisioning factor $\beta > 1$. The raw downstream bandwidth can then be obtained by multiplying β with the average bandwidth utilization shown in Table I. Simulations will use $\beta = 1.15$, which corresponds to slight over-provisioning and causes the downstream link to always have less bandwidth than the link where the traces were collected. The deque algorithm approximately overlays a logical link of bandwidth equal to $B/(1+r)$ for the fast queue traffic (Proposition 1). If the fast queue share is over-provisioned ($r < \beta - 1$), then the fast queue should be short, which is the primary objective of YAQS. The slow queue weight was set to $r = 0.1$, which corresponds to a slight over-provisioning of the fast queue bandwidth. The thresholds $\bar{c}_f$ and $\bar{c}_s$ were set to 16,500B in the initial set of experiments below. In general, this choice of parameters is meant to create difficulties for YAQS for the following reason. Smaller values of β and r can lead to longer queues and degrade service levels. Moreover, small values of $\bar{c}_f$ and $\bar{c}_s$ should force the classification algorithm to look for flows that have small queue occupancy relative to the long queue, and could increase the number of misclassifications of an approximate algorithm. In the following evaluation, approximate classification employs the Count-Sketch algorithm [21] with 3 hash tables of 32 buckets per hash table.

A classification algorithm should be fast and avoid maintaining per-flow state, but classification algorithms with no per-flow state are approximate and can occasionally misclassify flows. Accuracy can be expressed as the number of packets belonging to aggressive flows that are not individuated (false negatives) as well as the number of packets belonging to non-aggressive flows that are mistakenly classified as aggressive (false positives). In the parlance of Information Retrieval, the percentage of false negatives is a measure of the method's recall and the percentage of false positives is a measure of the method's precision. False negatives and false positives have different roles. False negatives are only relevant to the extent that they increase the fast queue length. On the other hand, false positives can critically worsen the QoS perceived by the victim flows.

Definition 3: Given an accuracy parameter $\epsilon \geq 0$, a *false positive* is a packet belonging to flow x for which either $\tilde{c}_f(x) > \bar{c}_f$ or $\tilde{c}_s(x) > \bar{c}_s$, and moreover $c_f(x) \leq (1-\epsilon)\bar{c}_f$ and $c_s(x) \leq (1-\epsilon)\bar{c}_s$, where $c_f(x)$ ($c_s(x)$) is the exact fast (slow) queue occupancy count of flow x at the time when the packet was enqued.

Definition 3 is inappropriate to quantify the dependency on $\bar{c}_s$ because $\tilde{c}_s$ is an exact count. Therefore, the definition is strengthened as

Definition 4: Given an accuracy parameter ϵ, a *strict false positive* is a packet belonging to flow x for which either $\tilde{c}_f(x) > \bar{c}_f$ or $\tilde{c}_s(x) > \bar{c}_s$, and moreover $c_f(x), c_s(x) \leq (1-\epsilon)\max\{\bar{c}_f, \bar{c}_s\}$.

Definition 4 is stricter than Definition 3 in that if a packet is a strict false positive, it is also a false positive, but the converse is not true. It also gives rise to a severe evaluation of YAQS in that the algorithm makes no attempt to promote a flow with $0 < \tilde{c}_s < \bar{c}_f$ (i.e., the algorithm does not attempt to remove strict false positives). If the number of misclassifications is small and sporadic, the main performance metric for YAQS is the queuing delay in the fast queue. The fast queue length can then be compared with the total queue length under other strategies, such as FIFO.

The main advantage of packet traces is that they capture behavior from actual links. Their main disadvantage is that traces are fixed and flows are not adaptive. Flow adaptiveness can be addressed by network simulations. Our simulation network is shown in Figure 2. We have explored the simulation sensitivity to all parameters but we report only on the most informative cases. The central link is the network bottleneck, has a bandwidth of 45Mbps and latency 1ms. The two endpoints hold 512 packets in each queue, and are YAQS, DropTail, and RED in our experiments. At each end, the link is attached to 14 routers by 10Mbps, 10ms links, and each of these routers is then connected to 6 end-nodes. Each end-node

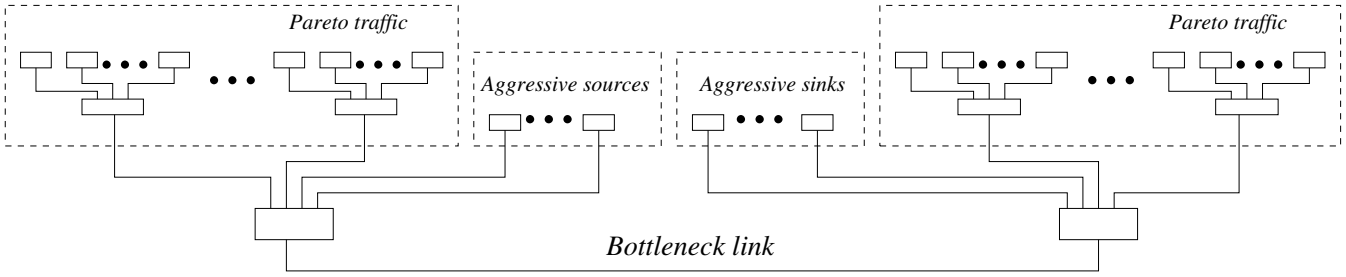

Fig. 2. Network used for simulations.

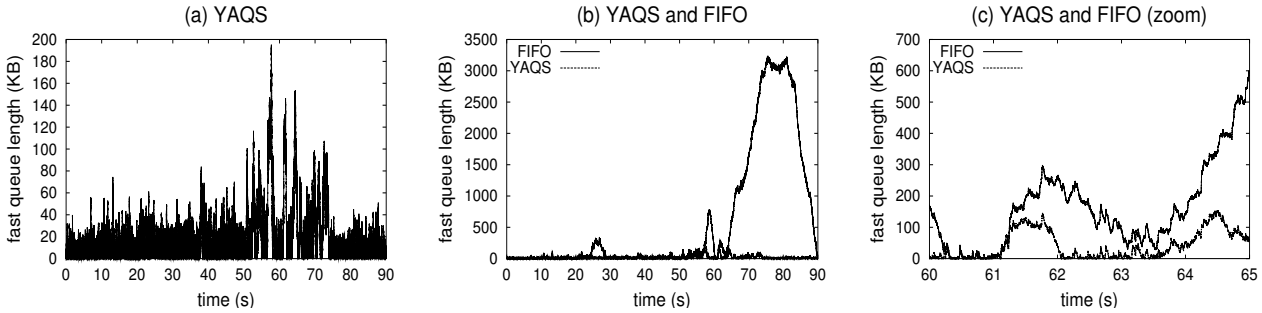

Fig. 3. (Fast) queue length (trace 6). Chart (a) gives YAQS fast queue length, chart (b) compares YAQS and FIFO, and chart (c) zooms on the interval $[60, 70]$, where the FIFO queue starts its highest ascent. The vertical axis scale differs among charts.

generates ON/OFF traffic following a Pareto distribution with shape equal to 1.2 and average file size of 8KB (an http-like distribution [4]). The Pareto traffic is then transported by TCP Sack/DelAck to a random end-node on the other side of the central link. Furthermore, average idle times are chosen so that the average traffic from all of these end-nodes is 14Mbps in both directions (i.e., the average utilization of the bottleneck bandwidth due to all these traffic sources is less than 1/3). Segment size for all TCP flows is 1460B, the initial window is 2 packets, and the advertised window is 1024 packets. In addition to these Pareto sources, the simulation employs n additional nodes that generate more aggressive flows (in the base case, $n = 1$). At time $1 + i/10$ seconds, node i will generate 4MB of data directed to a peer node on the other side of the central link ($0 \leq i < n$). All such "aggressive" nodes are connected to the central link by 100Mbps, 0.4ms links and (unlike the Pareto sources) all their data traffic flows from left to right. The aggressive RTT is relatively short so that, on the one hand, the TCP flows can quickly congest links and, on the other hand, AQM has a relatively faster dynamic on these flows. We have also experimented with the case when the aggressive sources are CBR/UDP flows, but the results are relatively uninteresting and omitted from the paper.

V. Evaluation

A. Trace 6

Summary data will be reported on all traces (Section V-B) but, for compactness, time-dependent charts will only be detailed for trace 6, where some of the longest queues form at the chosen value of β.

Figure 3 gives the fast queue length of YAQS and FIFO during trace 6. YAQS is a clear improvement over FIFO in terms of fast queue length. In fact, the YAQS line is but noise close to the horizontal axis when compared to FIFO. The comparison is better seen by zooming in, as in Figure 3(c). Figure 4(a) shows the complementary cumulative distribution of the fast queue length under YAQS. The probability of obtaining a long fast queue decreased exponentially. The corresponding distribution for FIFO in Figure 4(b) shows an excellent fit to a power law over three orders of magnitude. In other words, not only did YAQS shorten the queue length, but it also improved its qualitative behavior. Figure 5 reports on the number of flows in the slow queue. For comparison, the fast queue hosted up to 258 flows. In general, few flows are hosted in the slow queue and the distribution decays exponentially fast. Consequently, it is feasible to maintain exact occupancy counts $\tilde{c}_s$. The slow queue traffic contributed to 14% of the bytes that flowed through the link. Packet length inflation added around 279KB to the slow queue throughout the trace, which is equal to 1.6% of the non-inflated slow traffic. Most nop bytes (88%) are added when a flow is moved into an empty slow queue. Correspondingly, no flow was starved.

In the case $\epsilon = 1/2$, trace 6 had 18 false positives (Definition 3) out of 330,444 packets in the log. No flow had more than 3 false positives, although some of these multiple false positives were concentrated in a 1ms burst. The use of two counters $\tilde{c}_f$ and $\tilde{c}_s$ is critical for the accuracy of the classification algorithm. In an earlier version, we used only one counter $\tilde{c}$ to maintain the total flow occupancy in both

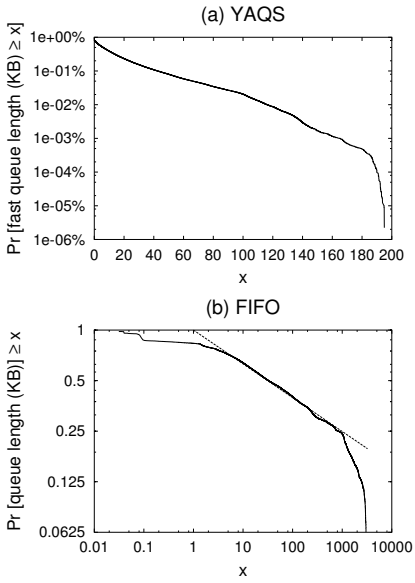

Fig. 4. Complementary cumulative distributions of the (fast) queue length (trace 6). Axis scale differs between charts.

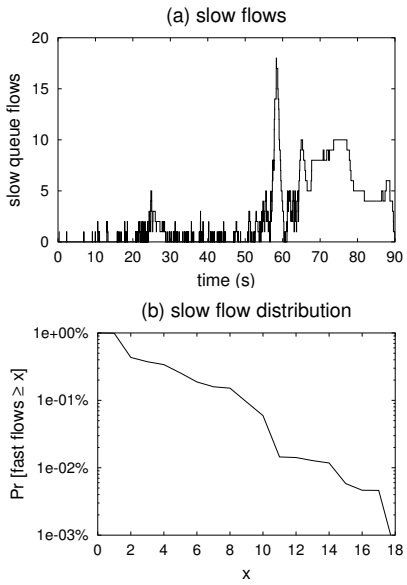

Fig. 5. Number of flows in the slow queue over time and its complementary cumulative distribution (trace 6).

the fast and the slow queue. The approximate counter does not maintain per-flow state and thus it lumps in the same tables statistics pertaining to flows with large and small queue occupancy. As a result, small occupancy flows were more often mistaken for flows with larger queue occupancy, i.e., false positives are more frequent. In the two counter scheme, $\tilde{c}_f$ maintains statistics for the fast queue, where flows should have small queue occupancy by design. Therefore, the classification algorithm has the easier job of identifying the larger queue occupancy outliers when they occur among a population of small occupancy flows. Another reason for the algorithm's accuracy is due to the use of exact occupancy counts for the slow queue (Line 2 in Algorithm 1): if an approximate estimate is used instead, the number of false positives increases to 434 and, more importantly, false positives are more concentrated either in time or by flows. Meanwhile, the small number of flows in the slow queue (Figure 5) even for this severe choice of β, $\bar{c}_s$, and $\bar{c}_f$ makes it feasible to efficiently maintain exact slow counts.

Another advantage of exactness in slow counts is that packet re-orderings can be completely eliminated if $\bar{c}_s = 0$. The value $\bar{c}_s = 0$ should be used, however, only if $\tilde{c}_s$ is exact, as approximate algorithms can accurately detect only the flows with larger occupancy values. The number of strict false positives (Definition 4) for $\bar{c}_s = 0$ and $\epsilon = 1/2$ was 606, which corresponds to 0.2% of the packets in the trace. Moreover, the smaller value $\bar{c}_s = 0$ led to shorter fast queues (Section V-B). It also led to a decrease in the maximum number of flows ever present simultaneously in the slow queue (no more than 10 flows were ever hosted in the slow queue at the same time). The reduction of slow flows at peak times appears to be due to the following dynamic behavior of the YAQS algorithm. The previous value $\bar{c}_s = 16500$ is quicker at releasing flows from the slow queue than $\bar{c}_s = 0$ is. Released flow move to the fast queue and share buffer space and bandwidth with the other fast flows. As a result, the fast queue length increases and the fast queue occupancy also increases for more fast flows, which are then all moved to the slow queue. The net result is that at the end of this process, more flows occupy the slow queue than in the case when the original flows were left in the slow queue throughout ($\bar{c}_s = 0$). Although $\bar{c}_s$ should be a parameter that an individual administrator could set, we feel that $\bar{c}_s = 0$ presents only a minor loss of accuracy compared to the benefit of completely removing all packet re-orderings. In the rest of the paper, $\bar{c}_s = 0$ will be assumed.

The findings above shed light also on YAQS's handling of aggregates. Between time 55s and 85s, up to 18 flows collude to congest the link bandwidth (Figure 5). However, all of these flows are individuated with excellent accuracy and then contained in the slow queue, with minor disruptions on the other flows (Figure 3).

B. Trace Summary

Summary data across all traces are discussed next. Figure 6 shows the maximum, average, and 90 percentile queue length. YAQS led to drastic reductions in the maximum queue length on the traces where FIFO had the longest queuing. Furthermore, the average queue length becomes more predictable: while the average FIFO queue length varied by 2 orders of magnitude, YAQS ranged between 8KB and 21KB. Figure 7 gives the maximum and average number of flows in the slow queue. The slow flow count was below 12 in all traces. The slow traffic accounted for 6% of the total number of bytes transmitted across all traces, and for no more than 20% on any single trace. Most nop bytes (77% to 96%) are added when a flow is moved into an empty slow queue, at which

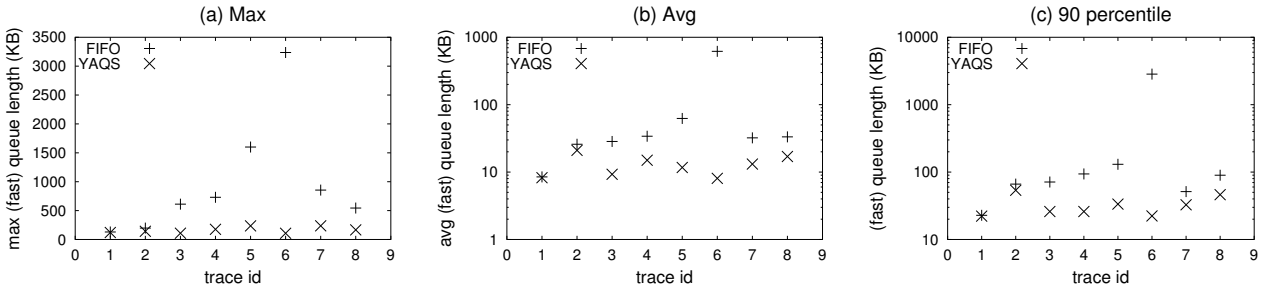

Fig. 6. Maximum, average, and 90 percentile queue length. The vertical axis scale differs among charts.

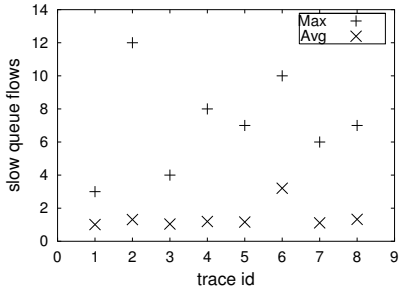

Fig. 7. Maximum and average number of flows in the slow queue.

point 2-4KB of nop traffic are added on average across the traces. The nop traffic was around 4% of the total slow traffic. The percentage of false positives never exceed 0.003% of logged packets across all traces and the percentage of strict false positive never exceeded 0.25%.

C. Simulations

As for simulations, Figure 8 gives the (fast) queue length of YAQS, FIFO, and RED during the activity of $n = 1$ aggressive flow. The thresholds are $\bar{c}_f = 16500$B, $\bar{c}_s = 0$B, and $r = 0.5$, which over-dimensions the fast queue bandwidth by roughly a factor of 2. In the case of RED queues, we used packet dropping and the default ns-2.1b9a parameters.

YAQS is a clear improvement over FIFO and RED in terms of fast queue length. In fact, the YAQS line looks once more like noise when compared to FIFO or RED. RED improves over DropTail, and it prevents long queues after an initial transient where it behaves like FIFO (i.e., until time 1.9s). However, RED did lead to long queues initially due to its slower dynamics, as discussed in Section II. (To compound the problem, the aggressive flow has a smaller RTT and consequently the benefit of faster dynamics than the Pareto flows.) The aggressive flow goodput is 24Mbps under YAQS, 29Mbps under FIFO, and 17Mbps under RED. In summary, YAQS was the most effective method to maintain short queues, and simultaneously the aggressive flow achieved higher goodput than under RED.

Figure 9 gives the queue length for $n = 1, 2, 4$ aggressive flows. The spikes in the fast queue length are due to various reasons. The most obvious cause is the initial slow start phase. Figure 10(a) details the initial period when each aggressive flow ramps up its queue occupancy and causes a separately identifiable spike. The fast queue subsequently shrinks when these flows are moved to the slow queue. Of course, similar spikes can occur after the initial slow start phases: Figure 10(b) shows the fast queue peak at time 2.8s caused by an aggressive flows that goes into slow start after having lost a retransmitted packet. Another reason for fast queue spikes is simply that less bandwidth is available to the fast flows when packets are waiting in the slow queue. For comparison with Figure 9, Figure 10(c) shows the FIFO queue length when there is no aggressive flow and the link bandwidth has been reduced to the $B/(1+r) = 30$Mbps fast bandwidth share. As for the aggressive goodput, it ranged from 7Mbps to 14Mbps across the 4 flows.

VI. Related Work

The local QoS paradigm can be instantiated in ways other than those described in this paper. For example, end-point applications or middleware can adapt to erratic network QoS [18]. The idea of treating flows differently depending on their behavior was developed in the context of AQM, for example, by FRED [19]. The RuN2C algorithm uses a two queue mechanism in order to achieve better response time for shorter flow, but it is based on TCP sequence numbers and does not consider instantaneous queue occupancy [27]. The significance of aggressive flows for real-time traffic stems from the measurement and characterization literature: first, it is shown that the largest traffic spikes across a link are due to sporadic flows that have relatively large end-to-end available bandwidth; second, it is demonstrated that the removal of such flows leads to excellent levels of service in an over-provisioned best-effort network [28], [29], [30], [31] (see also the measurements in [32]). A recent paper points out that the objective of QoS should be to minimize risk [33].

The GPS discipline can be approximated by various algorithms [34]. Algorithm 2 achieves the same fast T-WCI as the virtual clock algorithm [35] and it makes it easier than WF^{2}Q [26] to avoid re-orderings.

The problem of identifying frequently occurring items in a stream has received much attention. The earliest algorithmic work focused on the problem of finding an item that occurs at least half of the time [36], [37] or with a certain large

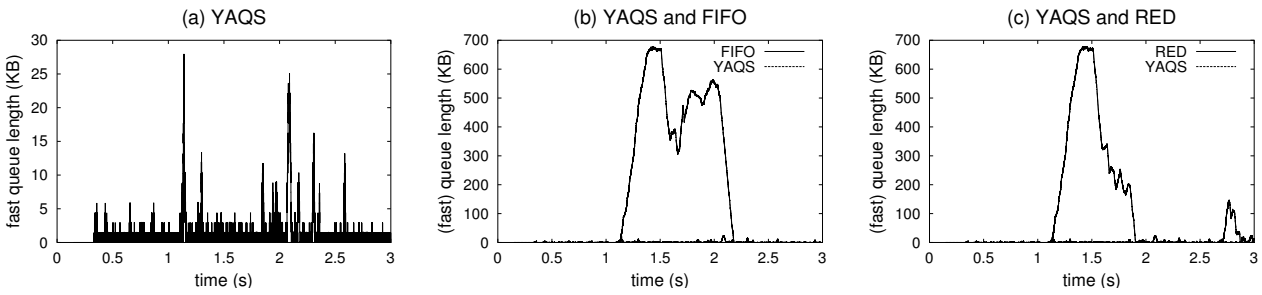

Fig. 8. (Fast) queue length in simulations. Chart (a) gives YAQS fast queue length, chart (b) compares YAQS and FIFO, and chart (c) compares YAQS with RED. The vertical axis scale differs between charts.

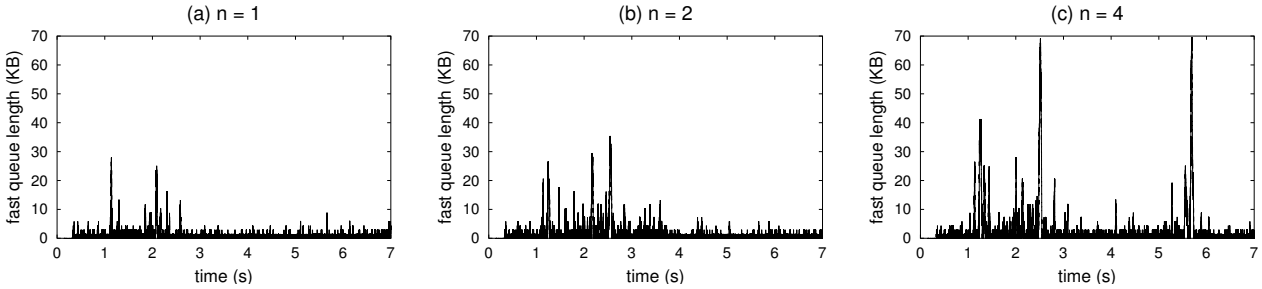

Fig. 9. Fast queue length in simulations with different numbers of aggressive flows.

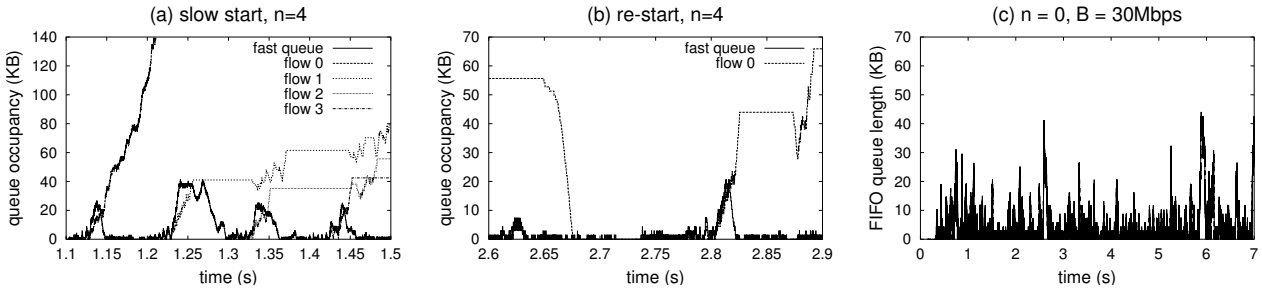

Fig. 10. Fast queue length in simulations and queue occupancy of the aggressive flows. The vertical axis scale differs among charts.

percentage [38]. A detailed background of this problem and its database and algorithmic ramifications can be found in [39]. Generally speaking, most of these approaches are suitable for massive data sets (e.g., a log of Google search queries, an off-line packet trace) but unsuitable to compute queue occupancy counters due to their high processing time, lack of support for packet departures, or both.

VII. Conclusions

The paper has introduced the YAQS queuing strategy. YAQS provides excellent protection against sporadic aggressive flows. YAQS individuates with high accuracy and little state the flows that occupy a large fraction of a buffer and segregates these offending flows into a separate slow queue. Extensive simulations show that other flows will benefit from short queuing delays while aggressive flows can still maintain high goodput. The analysis employed both ns simulations and packet traces collected on NLANR links.

A strategy such as YAQS is intermediate between two extremes: packets get differentiated handling at a router (as in explicit QoS) but a packet QoS class is ignored (as in best-effort). A direct consequence is that intermediate strategies, such as YAQS, can be selectively turned on or off at each router depending on the QoS needs at that link. At the core of the paper, the fundamental contribution is to elaborate on a local QoS paradigm that does not depend on multi-node cooperation. Local QoS is interoperable with best-effort networks, requires minimal router configuration but no multibox administration and no inter-provider coordination. In general, the local QoS paradigm has the potential to favor the incremental adoption of QoS in computer networks.

We feel that the paper poses more questions than answers. Future work should at least address end-to-end QoS on heterogeneous large-scale networks, a dynamic choice of the weight r, a comparison between YAQS and FQ, the interaction of

YAQS with AQM, and the queue behavior in the presence of shrew attacks [40] and aggregates.

ACKNOWLEDGMENTS

This work has been supported in part under NSF grants ANI-0123929 and CCR-0329910. NLANR traces are made made possible by the National Laboratory for Applied Network Research and NSF grants ANI-0129677 and ANI-9807479.

REFERENCES

[1] V. Cerf and R. Kahn, "A protocol for packet network interconnection," *IEEE Transactions on Communications Technology*, vol. COM-22, no. 5, pp. 627–641.

[2] C. Macian, L. Burgstahler, W. Payer, S. Junghans, C. Hauser, and J. Jaehnert, "Beyond technology: The missing pieces for QoS success," in *Workshop on Revisiting IP QoS (RIPQOS)*, 2003.

[3] A. Akella, S. Seshan, and A. Shaikh, "An empirical evaluation of wide-area Internet bottlenecks," in *Internet Measurement Conference*, 2003.

[4] B. Krishnamurthy and J. Rexford, *Web Protocols and Practice*. Boston: Addison-Wesley, 2001.

[5] P. K. Chrysanthis, V. Liberatore, and K. Pruhs, "Middleware support for multicast-based data dissemination: A working reality," in *Eighth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, 2003.

[6] H. Jamjoom and K. G. Shin, "Persistent dropping: An efficient control of traffic aggregates," in *Sigcomm*, 2003.

[7] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *Computer Communication Review*, vol. 32, no. 3, July 2002.

[8] S. Staniford, V. Paxson, and N. Weaver, "How to own the Internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium (Security '02)*, 2002.

[9] I. Norros, "On the use of Fractional Brownian Motion in the theory of connectionless networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 953–962, Aug. 1995.

[10] A. Manin and K. Ramakrishnan, "Gateway congestion control survey," RFC 1254, 1991.

[11] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks," in *SIGCOMM*, 1998, pp. 118–130.

[12] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[13] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED," 2001.

[14] H. Han, C. V. Hollot, Y. Chait, and V. Misra, "TCP networks stabilized by buffer-based AQMs," in *Infocom*, 2004.

[15] Z. Heying, L. Baohong, and D. Wenhua, "Design of a robust active queue management algorithm based on feedback compensation," in *Proc. Sigcomm*, 2003.

[16] C. V. Hollot, V. Misra, D. F. Towsley, and W. Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *INFOCOM*, 2001, pp. 1726–1734.

[17] P.-F. Quet, S. Chellappan, A. Durresi, M. Sridharan, H. Őzbay, and R. Jain, "Guidelines for optimizing multi-level ECN, using fluid flow based TCP model," in *Proceedings of ITCOM2002 Quality of Service over Next Generation Internet*, 2002, pp. 106–115.

[18] M. S. Branicky, V. Liberatore, and S. Phillips, "Co-simulation for co-design of networked control systems," in *American Control Conference*, 2003.

[19] D. Lin and R. Morris, "Dynamics of random early detection," in *SIGCOMM '97*, Cannes, France, september 1997, pp. 127–137. [Online]. Available: citeseer.nj.nec.com/lin97dynamics.html

[20] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe a stateless active queue management scheme for approximating fair bandwidth allocation," in *INFOCOM*, 2000.

[21] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Automata, Languages and Programming (ICALP)*, 2002, pp. 693–703.

[22] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," in *IEEE INFOCOM*, 2004.

[23] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "Stochastic fair blue: A queue management algorithm for enforcing fairness," in *INFOCOM*, 2001, pp. 1520–1529.

[24] A. Kumar, J. Xu, J. Wang, O. Spatscheck, and L. Li, "Space-code Bloom filter for efficient per-flow traffic measurement," in *Infocom*, 2004.

[25] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case." *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[26] J. C. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 675–689, Oct. 1997.

[27] K. Avrachenkov, U. Ayesta, P. Brown, and E. Nyberg, "Differentiation between short and long TCP flows: Predictability of the response time," in *Infocom*, 2004.

[28] S. Sarvotham, R. Riedi, and R. Baraniuk, "Connection-level analysis and modeling of network traffic," in *Proceedings IEEE/ACM SIGCOMM Internet Measurement Workshop*, 2001.

[29] S. Sarvotham, X. Wang, R. Riedi, and R. Baraniuk, "Additive and multiplicative mixture trees for network traffic modeling," in *Proceedings ICASSP*, 2002.

[30] S. Sarvotham, R. Riedi, and R. Baraniuk, "Connection-level analysis and modeling of network traffic," ECE Dept., Rice University, Tech. Rep., June 2001.

[31] X. Wang, S. Sarvotham, R. Riedi, and R. Baraniuk, "Connection-level modeling of network traffic," in *Proceedings DIMACS Workshop on Internet and WWW Measurement, Mapping and Modeling*, 2002.

[32] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling high-bandwidth flows at the congested router," in *Proc. ACM 9th International Conference on Network Protocols (ICNP)*, 2001.

[33] B. Teitelbaum and S. Shalunov, "What QoS research hasn't understood about risk," in *Workshop on Revisiting IP QoS (RIPQOS)*, 2003.

[34] H. Zhang, "Traffic control and QoS management in the Internet," in *ACM Sigmetrics Tutorial Proceedings*, 1999.

[35] L. Zhang, "Virtualclock: A new traffic control algorithm for packet-switched networks," *TOCS*, vol. 9, no. 2, pp. 101–124, 1991.

[36] B. Boyer and J. Moore, "A fast majority vote algorithm," Institute for Computer Science, University of Texas, Tech. Rep. 35, 1982.

[37] M. Fischer and S. Salzberg, "Finding a majority among n votes: Solution to problem 81-5," *Journal of Algorithms*, vol. 3, no. 4, pp. 376–379, 1982.

[38] J. Misra and D. Gries, "Finding repeated elements," *Science of Computer Programming*, vol. 2, pp. 143–152, 1982.

[39] G. Cormode and S. Muthukrishnan, "What's hot and what's not: Tracking frequent items dynamically," in *Proceedings of Principles of Database Systems*, 2003.

[40] A. Kuzmanovic and E. W. Knightly, "Low-rate TCP-targeted denial of service attacks (the shrew vs. the mice and elephants)," in *Proc. Sigcomm*, 2003.