Remote Active Queue Management

Dan Ardelean†, Ethan Blanton‡, Maxim Martynov‡
†Google, Inc.; ‡Purdue University
dardelean@google.com, {eblanton,martynov}@cs.purdue.edu

ABSTRACT

Consumers or administrators of small business networks usually cannot directly configure the link which connects their network to their Internet provider. The link setup often provides a sub-optimal configuration for end users' traffic patterns and, at best, favors bulk transfers. We introduce a method of imposing desired active queue management behavior on an upstream queue without requiring administrative control over the queue. We achieve this by observing various externally measurable characteristics of the queue's behavior and then manipulating congestion-controlled traffic through standard feedback channels such as packet drops or ECN notifications. This technique can be directly applied to improve the quality of VoIP connections sharing a bottleneck link with multiple TCP connections. Our approach performs comparably to a RED policy applied on the remote queue, without dropping VoIP packets. The complexity of our approach is independent of the number of connections, and hence it can be implemented on a network gateway without adding noticeable processing overhead.

1. INTRODUCTION

In small network setups, the network link connecting the end user's network to the Internet is usually not configurable from the downstream end. Given the network setup in Figure 1, where (H) is the downstream host, we manipulate the data stream received at (RD) to discipline the queue (Q) located at (RU) in an attempt to provide quality-of-service bounds to (H) and (P). In particular, we wish to bound the queuing delay an incoming packet sees in (Q) without adversely affecting the throughput of TCP connections traversing the (RU)-(RD) link, with the goal of improving link conditions for Voice over IP traffic. Providing a bound on queuing delay is especially important for interactive applications such as VoIP data streams, for which delays larger than about 250ms become noticeable in conversation.

To better understand this problem, we have created a number of simulations which assume that (Q) is a drop-tail queue with a longer queue than we desire, but consistent with conditions with which we are empirically familiar. We have additionally created simulations which approximate the desired behavior in terms of queuing delay; these simulations use the same topologies and data connections as the previous set, but with (Q) an appropriately tuned RED [4] queue. This is an effective lower bound for the delay and jitter we could expect to accomplish by disciplining (Q) from (RD). We then introduce a set of algorithms to be deployed at (RD) which closely mimic the behavior of a RED queue at (Q), and which can be easily deployed in consumer-grade DSL or cable routers.

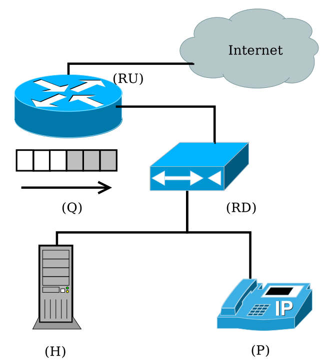

Figure 1: A small network layout

Throughout this work we assume that the network bottleneck is located at the Internet access link on the receiver side, which is typically the case for small office networks, often connected to the Internet via low-bandwidth DSL or cable. Thus, we are concerned only with congestion of the downstream link. While it is possible for the upstream link to become a bottleneck, we assume that the sender has full administrative control over the congested queue in that case, and can deploy an appropriate queue management technique.

The remainder of this paper is structured as follows: Section 2 describes at a high level our approach to solving this problem. Section 3 describes a number of queue estimation techniques, some of which proved fruitful and some of which were proven (or empirically shown) unworkable. Section 4 describes our proposed algorithm, and Section 5 presents the results of some of the simulations demonstrating the problem and the efficacy of our solutions. Section 6 discusses our conclusions and final results.

2. BASIC APPROACH

Providing a bound on the depth of (Q) from (RD) (re-

NOSSDAV '08 Braunschweig, Germany

fer to Figure 1) implies that (RD) will have to indirectly deduce the state of (Q). (RU) and (RD) exist in different administrative domains, and therefore tools such as SNMP are not likely to be available. In consequence, we will have to use externally detectable characteristics of (Q) (such as propagation delay and packet drops) to learn about its state.

Since our primary focus is the mitigation of delay and jitter for VoIP streams, we have chosen to express the depth of the queue (Q) in terms of delay. In most cases this is directly proportional to depth in bytes; it may diverge somewhat from depth in packets in the face of widely varying packet sizes, or it may be independent of packet sizes but directly proportional to the depth in packets in queues from which packets are clocked at regular intervals. Our mitigation technique is therefore to estimate the queuing delay of (Q) and use this estimation to inform a probabilistic dropping algorithm in the spirit of RED. As our target platform is consumer-grade set-top network elements, we are restricting the algorithms to those which can be computed relatively efficiently and in a constant amount of space — independent of the number of connections traversing the link or the bandwidth of the link itself. This rules out more stateful schemes such as [10], [8], [12] and [9], which apply receiver-based TCP window control, and, hence require keeping state information for each TCP connection.

Note that our scheme is dropping inbound packets at (RD), which means that drops are "expensive" — in the sense that any packet dropped inbound at (RD) has already traversed the link (RU)-(RD), which is presumed to be the bottleneck on our path. The dropped packet directly translates to lost bandwidth on that link. Note that the dropping of pure ACK packets, while avoiding data loss, will not achieve the desired effect, as this will not cause TCP congestion response mechanisms to be triggered.

Our main goal in this approach is to enforce the queuing policy *remotely*, and to approximate direct control over the remote queue as closely as possible. That is, we are not devising a queuing policy to compete with existing policies, but rather proposing a way to force a remote queue to follow some existing policy, such as RED.

3. QUEUE ESTIMATION TECHNIQUES

We have tried several techniques for estimating the queue depth at (Q), both experimentally and theoretically. The ideal scheme would closely estimate the queue length without phase shift, would be passive, and would make no specific assumptions about the underlying traffic. We emphasize that it is the current queue depth that we need to estimate, and not the parameters of individual connections which share the queue.

3.1 Rate Based Estimation

This technique tries to determine the queue length based on the observed rate at (RD). The estimation assumes traffic to be congestion controlled and relies on a model that assumes that the amount of traffic in congestion avoidance (as opposed to slow start) is a constant share of the total traffic. Furthermore, the estimation assumes that the rate of the congestion avoidance (CA) traffic behaves linearly, while the rate of the slow start traffic behaves exponentially. The assumptions are fomalized as follows:

$$R(t+\Delta t) \;=\; \alpha(R(t) \;+\; A\Delta t) \;+\; (1 \;-\; \alpha)R(t)2^{B\Delta t}$$

where $R(t)$ is the total rate at time t, Δt is a small time interval, α is the share of congestion avoidance traffic, A is the linear coefficient and B is the exponential coefficient. The equation is solved for $R(t)$ as

$$R(t) \;=\; R_0 2^{(1-\alpha)Bt} \;=\; R_0 2^{Ct}$$

where R_0 is the initial rate and C is constant. Then $C = \frac{dR(t)}{dt} \cdot \frac{1}{R(t)\ln\ 2}$. $\frac{dR(t)}{dt}$ can be obtained from a discrete approximation, which allows to calculate C and predict future values of $R(t)$. The queue depth, in turn, is updated at each step of rate estimation as

$$Q(t_i) = Q(t_{i-1}) + (R(t_{i-1}) - r)(t_i - t_{i-1})$$

Here r is essentially a bottlneck link speed. The estimated queue depth is used to apply some queue management policy, such as RED, to all incoming packets, as if we had a local queue of such depth.

This method proved to estimate rate (and queue length) fairly well when no packet drops occur at (RD). Predicting the effects that a random packet drop has on the queue length variation is problematic, as demonstrated by the result in Figure 2. The result was obtained with a simulation of multiple TCP flows sharing an access link. It can be observed that the queue depth is estimated accurately till the first packet drop, but then the difference between the estimated and the true queue depths leads to the queue being completely empty several times, which causes the link to be underutilized. We have not yet found a good way to estimate the dropping effects in this scheme without keeping complicated state.

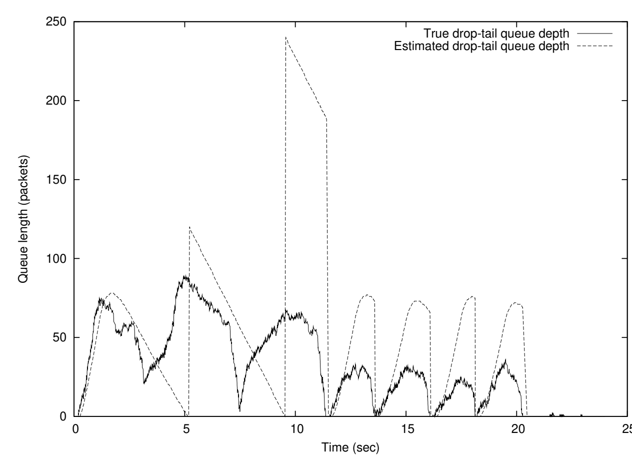

Figure 2: Rate-based Queue Depth Estimation

3.2 TCP RTT Estimation

Taking measurements on the traffic we want to manipulate in order to control delay and feedback is convenient because it is by definition present when the manipulation takes place. Because TCP generates acknowledgment packets in reply to specific data segments, it is possible to take timings of the data packet and its returning ACK to establish a round trip time measurement. Then the RTT variation can be used to derive information about queuing delay. Previous research [6] has shown that careful sampling may make it possible to retrieve measurements with sufficient precision for our purposes under the right circumstances.

If we assume that RTT variation at the current time t, $\delta(t)$, is due to queuing delay on the access link, and we know the rate r at which the queue is drained, we may estimate the current queue depth as $r\delta(t)$, and then apply a queue management policy as if we had a local queue of such depth.

There are several hurdles to be overcome for this technique; for example, the delayed ACK timer [2] can extend the measured round trip time for acknowledgment of arbitrary data segments for many milliseconds. However, storing timings for two data segments back-to-back would meet our requirements of constant state while providing timings with reasonable granularity for steady-state connections (*i.e.*, allowing us to identify ACKs triggered by a received data segment, rather than the delayed ACK timer). Additionally, ACK packets are not reliable and may be lost with some frequency without greatly affecting the TCP connection, making our measurements difficult. Finally, when using ACKs for RTT estimation, we require a presence of some data that is going upstream via open TCP connections, but such data flow might not be available.

This work does not include a passive TCP RTT estimator, but this direction should be explored.

3.3 Real-time Stream Based Estimation

Searching for a passive and effective way to determine the remote queuing delay, it became clear that we need more information than we can derive by monitoring only specific types of traffic which are not both highly periodic and frequent. The VoIP traffic for which we are trying to improve latency already has these properties: packets carrying the VoIP stream are normally sent at regular intervals and are by definition present when the queue estimation is needed; they also contain a timestamp [11, 5].

We do not make any assumptions about the relationship of these timestamps to wall clock time, nor about the synchronization between (RD) and the remote VoIP stream sender's clocks. Therefore, these timestamps cannot be used as a direct indication of end-to-end queuing and transmission delay. However, by arbitrarily declaring the "fastest" of the first few packets we see (based on their inter-arrival times) as having a nominal queuing delay and then adjusting the perceived delay upward and downward based on the deltas between subsequent packet pairs, we arrive at an estimation of the depth of the queue at (RU). To accomplish this, we maintain a heartbeat timer on (RD) synchronized to the expected packet arrival times. At each packet arrival we calculate the difference between its expected and actual arrival time. Packets arriving earlier than the expected arrival time are assumed to indicate that our initial, arbitrary measurement was in fact *not* nominal, and the heartbeat is adjusted backward to compensate. Packets arriving later than the expected arrival time are assumed to represent queuing delay at (Q). This estimation is further improved by taking advantage of the arrival of non-VoIP stream packets; when a non-VoIP stream packet is received after the arrival of a VoIP stream packet was expected, the delay timer is adjusted accordingly — this allows interleaved competing traffic to incrementally adjust the perceived delay while pending VoIP stream arrivals remain in the queue.

The results of a sample run of this delay estimation are shown in Figure 3. In our simulations, we use the IAX VoIP protocol [11] due to its simplicity and because close fit to our VoIP stream model (packets are highly periodic and timestamped). The graph shows the actual and estimated queuing delays at (Q) for an experiment involving several HTTP connections simulated using the model in [3] and one real-time VoIP stream. The temporal shift between the actual delay and the estimated delay function is an artifact of the link delay itself, as the estimation can only be adjusted once a given packet has fully traversed the link; one can compensate for this shift *a posteriori*, but in real-time queue manipulation, its effects are unavoidable. For clarity, Figure 3 shows only five seconds of the simulation; the remainder of the 25-second simulation shows the same characteristics.

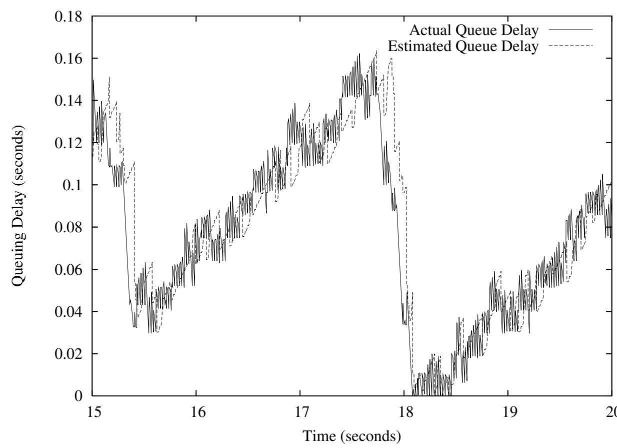

Figure 3: Delay Estimation

This scheme produces estimations that closely follow the actual delay of the queue in (RU). The trade-off is that it relies on application specific data.

4. REMOTE ACTIVE QUEUE MANAGEMENT

The Remote Active Queue Management (RAQM) process on (RD) uses the estimated queuing delay of (Q), as computed by one of the heuristics in Section 3, to inform packet dropping at (RD) or ECN [7] marking on the outbound queue of the (RD)-(RU) link. In our simulations, we use the real-time stream based estimation technique in conjunction with a RED-like probabilistic packet dropping. When ECN-capable traffic is available, one could mark packets with the ECN Congestion Experienced bit in order to achieve the same effect on (Q) without losing the contents of the marked packets; as this should have the same effect as packet dropping and lead to strictly better performance figures for our queue, we chose to use packet dropping in order to establish a pessimistic performance bound.

Traditional RED algorithms perform their marking and dropping based not on an instantaneous queue size, but on a smoothed average of recent queue sizes. Our algorithm, on the other hand, uses a more naïve instantaneous delay in calculating its dropping probability. Simulations show that RED's smoothed average can work well on the provider's queue; however, we had little success in using a smoothed delay estimation, and found that for sensitive links an instantaneous estimation yielded better results. This remains an area open for future work.

Representing the last instantaneous delay measurement as d, the configurable minimum delay threshold (below which packets will not be marked or dropped) of the RAQM queue as d_{min}, and the maximum delay threshold (above which all packets will have maximum drop probability) as d_{max}, we calculate the drop probability p between the minimum drop probability p_{min} and the maximum drop probability p_{max} as follows. (Note that d_{min} and d_{max} are analogous to RED's min_{th} and max_{th}. Both are constant, configurable parameters, independent of instantaneous delay measurements.)

$$p = \max(0.0, \min(1.0, \frac{d - d_{min}}{d_{max} - d_{min}})) \cdot (p_{max} - p_{min}) + p_{min}$$

Real-time packets from the multimedia streams we are trying to improve are not dropped by this algorithm.

The queue implementation used in this work assumes that the (RU)-(RD) link is the bottleneck link for the majority of the congestion-controlled connections comprising the traffic in the (RU)-(RD) direction. Furthermore, it requires that traffic in the (RD)-(RU) direction does not build a queue at (RD); this can be accomplished by using standard active queue management at (RD). Compensation for small queuing delays incurred at (RD) can be worked into the algorithm, but as no queue builds for topologies and traffic patterns in this work, we implemented no such correction.

5. SIMULATIONS

In this section we are evaluating the RAQM approach by comparing it to Drop-tail and RED policies deployed directly on the congested queue. Again we emphasize that we do not need RAQM to outperform RED, but rather approximate it, as if RED were being used to control the remote queue.

5.1 Extensions to the `ns-2` Simulator

We introduced an implementation of the RAQM queue to the `ns-2` [13] simulator, which uses the real-time stream estimator described in Section 3.3 in conjunction with simulated IAX protocol traffic. Using these extensions, we constructed a network topology in which (RD) (from Figure 1) has an inbound RAQM queue on the (RU)-(RD) link.

For each of the following simulations, our target queuing delay for (Q) is no more than 250ms. Additionally, we consider any packet which arrives more than 250ms "late" to be unusable. These numbers were selected based on desirable end-to-end characteristics for VoIP connections.

5.2 A Simple Topology

We performed a number of experiments on a classic "dumbbell" topology (similar to Figure 1), with (H) connected to (RD) by a fast 10Mbps low-delay connection and two remote hosts connected to (RU) by a 10Mbps link with 50ms propagation delay. The bottleneck link connecting the two ends of the dumbbell was configured to approximate a consumer ADSL connection, as an asymmetric 768Kbps/128Kbps link with 3ms downstream and 10ms upstream one-way propagation delays. There are two competing TCP connections terminating at (H) crossing the bottleneck link (one from each remote host) as well as a single IAX flow, all of which start at time 0 and run through the end of the simulation, 50 seconds later.

For this scenario, we first looked at characteristics of the IAX stream. We considered the mean delay and standard deviation from this delay of each IAX packet, the number of IAX packets dropped from (Q) (out of 2490 crossing the bottleneck link), and the number of "usable" IAX packets received (those experiencing a delay of $\leq$ 250ms). These statistics are presented in Table 1.

Queue	Mean Delay	Delay SD	Drops	Usable
Drop-Tail	421ms	124ms	9	264
RED	105ms	52ms	5	2471
RAQM	83ms	48ms	0	2463

Table 1: IAX Statistics for the Simple Topology

Queue	Utilization	Goodput	Fairness
Drop-Tail	99.4%	4344	2%
RED	99.0%	4335	3%
RAQM	99.1%	4274	13%

Table 2: TCP Statistics for the Simple Topology

As shown by Table 1, the RAQM Queue provides a better VoIP experience than the unmitigated drop-tail queue. While only 10% of the IAX packets in the drop-tail scenario arrive within our designated queuing delay of 250ms, the RAQM link delivers 99% of the IAX packets in time, performing comparably to the RED link. Notice that the number of drops shown in Table 1 is the number of dropped IAX packets only, not the total number of dropped packets over all connections. The table, thus, demonstrates this technique reduces the packet loss rate for VoIP.

We next looked at the impact the various queuing schemes had on the general link characteristics. We considered overall bottleneck link utilization, TCP goodput (defined as the amount of TCP data successfully transferred over the link expressed in MSS-sized segments), and the "fairness" among the two connections expressed as the percentage of difference in their goodputs (smaller is better). Table 2 indicates that the overall link utilization is comparable to both "traditional" queues under the RAQM scheme, though fairness suffers slightly. This fairness difference appears to be due to more bursty drop patterns exhibited by the RAQM algorithm, stemming from the fact that its moving average of link delay is much more sensitive than, e.g., the RED average queue length, without the mitigating effect of a very deep queue provided by the drop-tail link. The similar link utilization but poorer goodput for RAQM compared to RED is explained by the fact that RAQM's drops are performed on the (RD) side of the link — RAQM dropped 48 packets in this simulation, losing the bandwidth for those packets despite their significance in link utilization.

5.3 A More Complex Topology

For the next set of evaluations we considered the somewhat more complex topology shown in Figure 4. The links for each (S_i) were chosen to represent typical network paths, based on informal latency measurements conducted from a small number of consumer broadband connections. They represent, from (S_1) to (S_5) respectively, a search engine, a content portal, a consumer DSL connection, a news site, and a blog facility. Each (S_i) was chosen as a popular example of its type of site. The (RU)-(RD) link is again a 768Kbps downlink/128Kbps uplink ADSL connection. The IAX connection in this scenario is between (H) and (S_3), the remote consumer DSL host.

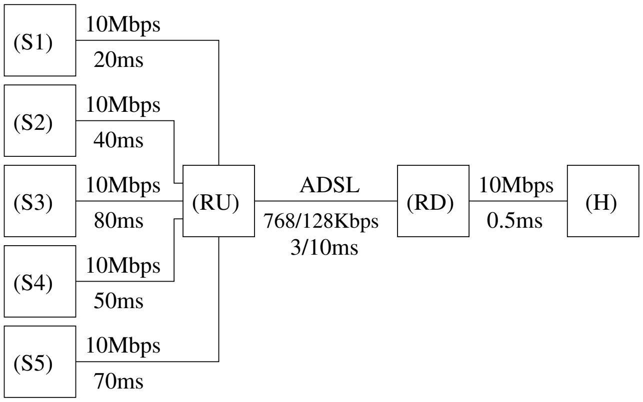

Figure 4: Complex network layout

Queue	Mean Delay	Delay SD	Drops	Usable
Drop-Tail	598ms	111ms	17	57
RED	104ms	40ms	16	2463
RAQM	118ms	46ms	0	2450

Table 3: IAX Statistics for the Complex Topology

Table 3 summarizes the same set of statistics as Table 1 for this new topology.

Table 4, analogous to the simple topology's Table 2, shows that this great improvement of the RAQM-monitored link over the drop-tail link in real-time performance comes at only a mild penalty in link utilization and slight loss of goodput. This loss of goodput for RAQM is again explained by the fact that its dropping takes place at the (RD) end of the "expensive" link. In this case, the total penalty in comparison with the RED queue is about 3%. Fairness is not evaluated here due to the complexity of comparing differing end-to-end paths, but the RAQM and RED queues achieve similar total goodput from each respective (S_i).

Figure 5 illustrates that the gross behavior patterns of RED and the RAQM queue are very similar. The delay spike at the beginning of the simulation is caused by multiple TCP flows entering slow-start simultaneously, a scenario which does not happen very often, but one which is difficult to handle for both RED and RAQM. For clarity, Figure 5 shows only 20 seconds of the simulation; the remainder of the 50-second simulation shows the same characteristics.

5.4 Simulated Web Traffic

We used the PackMime[3] realistic traffic generation package for `ns` to generate a more realistic pattern of TCP connections in order to test our algorithm in a more dynamic environment. This traffic generation was used with the "complex" topology described in section 5.3 in conjunction with

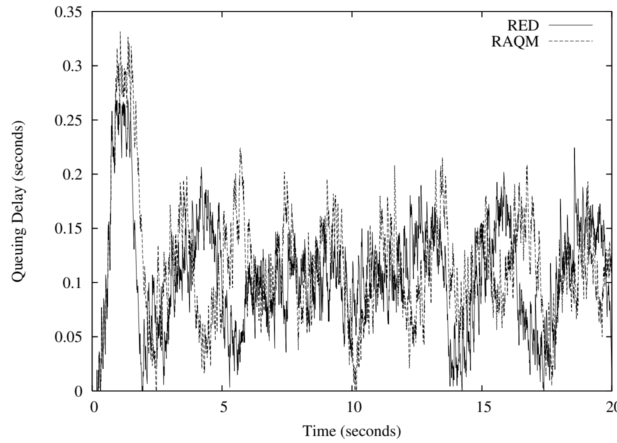

Figure 5: Queuing delay with RED and RAQM Queues

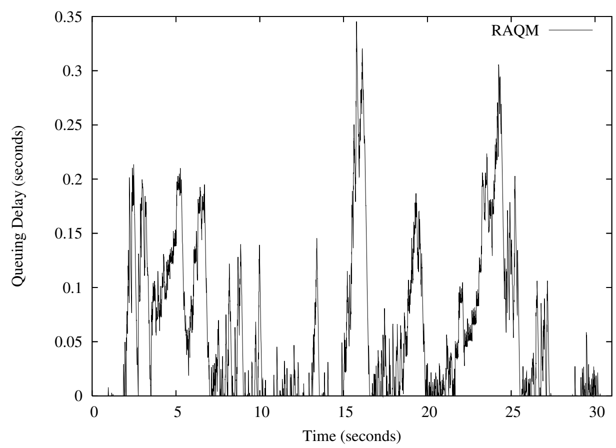

Figure 6: RAQM Queuing delay with a Web Traffic Mix

each of the three queue types. Each host (S_i) was configured as a "server" for (H), with (H) making at most four connections per second per server in a pattern consistent with fetching a web page and its associated graphics and other objects (of varying sizes) for fifteen seconds. The simulation was allowed to run for 45 seconds, allowing time for all connections to complete. Fetch completion took roughly 30.3 seconds for the drop-tail and RED queues, and 32.5 seconds for the RAQM queue.

Link utilization statistics are not as useful in this scenario as the previous, due to the fact that there are natural idle periods between page fetches. This scenario is particularly interesting because it involves connections in both slow start

Queue	Utilization	Goodput
Drop-Tail	99.7%	4379
RED	99.7%	4383
RAQM	97.7%	4237

Table 4: TCP Statistics for the Complex Topology

Queue	Mean Delay	Delay SD	Drops	Usable
Drop-Tail	117ms	182ms	0	1680
RED	37ms	62ms	19	2201
RAQM	39ms	64ms	0	2215

Table 5: IAX Statistics for the Complex Topology

and congestion avoidance, with new connections intruding on a "stable" network. Figure 6 shows the delay plot for the RAQM queue and simulated web traffic.

Table 5 gives some statistics about the quality of service provided to the IAX connections with web traffic; there were a total of 2246 IAX packets transmitted in these scenarios. Note that neither the RED nor RAQM scenarios lose more than a few consecutive samples (according to our 250ms "usability" metric), but the drop-tail scenario, in addition to having a poor overall rate of unusable packets, loses about 5 contiguous seconds of samples at two different points during the simulation.

6. CONCLUSIONS AND FUTURE WORK

This work presents a Remote Active Queue Management algorithm which can be used to bound the queuing delay packets experience in a queue upstream of a RAQM-enabled router. The problem RAQM solves often arises when ISPs configure deep queues on the remote side of a slow link in order to improve total throughput and smooth out bursts. However, real-time streams which require delay guarantees are not helped by these deep queues, as even when real-time packets do not experience loss, they may become useless due to accumulated delay. As opposed to the techniques proposed in [10], [12], [9] and [1], we try to improve the QoS of real-time streams at one end of the slow link instead of at end hosts. Furthermore, our scheme is not as intrusive as [1] in the sense that it does not require the total replacement of the IP stack, and scales well with the number of TCP connections in contrast to the schemes proposed in [10, 12, 9], which keep track of individual TCP connections.

RAQM is based on estimating the queue length on the remote end and selectively dropping packets belonging to congestion controlled streams, while maintaining fairness and efficiently bounding the delay. The algorithm requires only a constant amount of state and a simple per-packet computation, making it appropriate for deployment on severely resource-constrained consumer devices. Additionally, various techniques for estimating queue length are presented, and a novel method for using periodic real-time packets to track the queuing delay at an upstream queue is implemented and evaluated. Proof-of-concept simulations establish that the RAQM algorithm can be successfully used in a number of situations to improve latency characteristics.

We found that we could cause RAQM to closely approximate an upstream RED queue of a given behavior with some tweaking of the RAQM configuration parameters. Low-bandwidth higher-delay links remain problematic (such as represented by, for example, ISDN or very slow DSL connections) for the RAQM algorithm, forcing a choice between fairness and delay quality in some circumstances. However, even moderately faster links such as the ones examined in this paper proved much more manageable and showed promising preliminary results.

While RAQM as described in this document seems to be functional under many circumstances, there remain many questions open for future work. For example:

- This work does not consider "turning off" RAQM, such as in the absence of real-time flows. Doing so, as well as rapidly turning RAQM monitoring back on at the arrival of a real-time flow, remains open for future study.
- Identifying and evaluating additional, and perhaps more flexible, estimation algorithms would broaden the applicability of RAQM and provide for rapid response at real-time flow arrival.
- There is room for improvement of RAQM behavior on slow, long-delay links, as well as in situations where multiple TCP flows enter slow start at the same time.
- The interaction of multiple RAQM flows in an end-to-end path should be explored, as well as the possibility of compensating for flows which have differing bottleneck delays.

Acknowledgments

We would like to thank Mark Allman for his extensive comments on this work, and Sonia Fahmy for support and direction during the early stages of development. We also thank Douglas Comer for his helpful comments.

7. REFERENCES

[1] H. Balakrishnan and S. Seshan. The congestion manager. RFC 3124, 2001.

[2] R. Braden. Requirements for internet hosts – communication layers. RFC 1122, 1989.

[3] J. Cao, W. Cleveland, Y. Gao, K. Jeffay, F. Smith, and M. Weigle. Stochastic Models for Generating Synthetic HTTP Source Traffic. In *IEEE INFOCOM*, 2004.

[4] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, Aug. 1993.

[5] R. F. H. Schulzrinne, S. Casner and V. Jacobson. RTP: A transport protocol for real-time applications. RFC 3550, 2003.

[6] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *SIGCOMM Comput. Commun. Rev.*, 32(3):75–88, 2002.

[7] S. F. K. Ramakrishnan and D. Black. The addition of explicit congestion notification (ECN) to IP. RFC 3168, 2001.

[8] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Explicit window adaptation: A method to enhance TCP performance. *IEEE/ACM Trans. Netw.*, 10(3), 2002.

[9] P. Key, L. Massoulié, and B. Wang. Emulating low-priority transport at the application layer: a background transfer service. In *ACM SIGMETRICS/Performance*, pages 118–129, 2004.

[10] P. Mehra, A. Zakhor, and C. De Vleeschouwer. Receiver-driven bandwidth sharing for TCP. In *IEEE INFOCOM*, 2003.

[11] M. Spencer, B. Capouch, E. Guy, F. Millera, and K. Shumard. IAX2: Inter-Asterisk eXchange Version 2, 2006. IETF Internet-Draft.

[12] N. T. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. N. Bershad. Receiver based management of low bandwidth access links. In *IEEE INFOCOM*, pages 245–254, 2000.

[13] Various. The Network Simulator — ns-2. `http://www.isi.edu/nsnam/ns/index.html`.