PURPLE: Predictive Active Queue Management Utilizing Congestion Information

Roman Pletka Marcel Waldvogel
IBM Research
Zurich Research Laboratory
{rap,mwl}@zurich.ibm.com

Soenke Mannal*
Institute for System Dynamics and Control Engineering
University of Stuttgart
soenke.mannal@isr.uni-stuttgart.de

***Abstract*— Active Queue Management (AQM) tries to find a delicate balance between two antagonistic Internet queuing requirements: First, buffer space should be maximized to accommodate the possibly huge transient bursts; second, buffer occupation should be minimum so as not to introduce unnecessary end-to-end delays. Traditional AQM mechanisms have been built on heuristics to achieve this balance, and have mostly done so quite well, but often require manual tuning or resulted in slow convergence. In contrast, the PURPLE approach predicts the impact of its own actions on the behavior of reactive protocols and thus on the short-term future traffic without per-flow state. PURPLE allows much faster convergence of the main AQM parameters, at least towards a local optimum, thereby smoothing and minimizing both congestion feedback and queue occupancy. To improve the quality of the prediction, we also passively monitor (using lightweight operations) information pertaining to the amount of congestion elsewhere in the network, for example, as seen by flows traversing this router.**

I. Introduction

The predominant transport protocol in the Internet is TCP, which provides reliable delivery and response to congestion. TCP, when tasked with bulk transfers, will slowly but steadily increase the transmission rate until it is notified of congestion, whereupon it drastically reduces the transmission rate, only to repeat the entire process [1]. The congestion indication is traditionally delivered through packet loss, which is in turn controlled by the router's queuing policy.

Queuing policy in the Internet is governed by two antagonistic requirements: First, buffer space should be maximized to accommodate sometimes huge transient bursts; often, sufficient memory is provided to buffer an entire end-to-end round-trip time's (RTT) worth of link bandwidth [2]. Second, buffer occupation should be minimum so as not to introduce unnecessary end-to-end delays: if each of the often as many as 30 routers between source and destination hosts were to buffer 200 ms of traffic in both directions, each message would require 6 s round trip, which clearly is not realistic.

Even though this extreme scenario is unlikely to ever happen, it shows the importance of maintaining a low average queue occupancy while allowing for large peaks. Several requirements are coupled to end-to-end packet delivery, that can be influenced by queuing mechanisms, and that ultimately are all interrelated:

1) *Low packet-loss rates.*
2) *Short end-to-end delays,* by controlling the contribution of queuing delay.
3) *High goodput* is achieved by reducing the number of retransmitted packets, which in turn cause additional high end-to-end delays. Goodput is defined as the effective data rate observed by the application.
4) *Ability to absorb bursts,* which requires sufficient headroom.
5) *Stable queuing delays* are beneficial for real-time applications such as voice transmission as well as for reliable transport protocols, which need to be able to estimate retransmission timeouts.

The traditional method of FIFO queues with tail-drop on queue overflow do not fulfill these requirements very well, because queues tend to oscillate between empty and full. Therefore, Active Queue Management (AQM) was conceived to improve performance of these metrics and to improve interaction with TCP's congestion control. TCP, like most other protocols in the Internet that respond to congestion, takes packet loss as an indication of network congestion. To keep queue occupancy low, most instances of AQMs start dropping packets (therewith indicating congestion) with low probability already at small queue occupancy levels, and gradually increase the congestion signal as the queue grows. This causes the responsive protocols to reduce their transmission rate and thus eventually reduce the queuing delay. Another advantage of this approach is that losses no longer occur in bursts, resulting in faster recovery from transmission losses.

Despite the importance of AQM, the actual mechanisms proposed have mostly been built on heuristics to achieve their goals. Even though most AQMs significantly improve on these goals, they often need manual tuning or provide only slow convergence [3]–[5].

The recent introduction of Explicit Congestion Notification (ECN) [6] has opened new avenues. Without dropping packets and thus causing a later retransmit or even a time-out, it is now possible to indicate congestion to participating transmission protocols by setting a mark in the packet header. This has the potential to significantly improve goodput, but current AQM mechanisms such as the industry standard RED [7] often need to revert to packet loss.

The PURPLE approach, in contrast, predicts the impact of

*Work done while at IBM Research, Zurich Research Laboratory.

its own actions on the behavior of reactive protocols and thus the short-term future traffic. PURPLE achieves this by analyzing end-to-end information about the congestion state in the network. PURPLE allows much faster convergence of the main AQM parameters, at least towards a local optimum, thereby smoothing and minimizing both congestion feedback and queue occupancy. To improve the prediction, we also passively monitor (using lightweight operations) information pertaining to the amount of congestion elsewhere in the network as seen by flows traversing this router.

A. *Our Contribution*

The design of PURPLE introduces three powerful new tools into AQM research. First, we extend the current myopic view of routers, which only consider the links they manage. Instead, we extract end-to-end congestion information pertinent to the packet flows traversing the managed links. This is done by simply counting the ECN marks in the traversing packets, which does not require any per-flow state. From this we derive how much congestion has already been indicated by previous routers and estimate how much congestion will be introduced after the packets leave the router. According, this router can fine-tune its contribution much more accurately.

Second, we are able to estimate flow RTTs by monitoring ECN information embedded in packets, without requiring knowledge of the detailed transport protocol semantics.

Third, we refrain from resorting to pure heuristics for AQM and try to provide a solid foundation by controlling the traffic stream using optimized parameters derived online from model-based predictions. By using the extracted information, it is possible to use the TCP steady-state model [1], [8] to predict the reaction of the end systems and thus provide a more accurate regulation of the control signal. Even though not all TCP flows are governed by the steady-state equation, it appears to be a good model even for the other flows.

Finally, we demonstrate the successful integration of these three new tools into a strong new AQM mechanism and compare PURPLE with other AQM schemes by means of simulations. PURPLE significantly improves on the performance of the best-known AQM mechanisms such as RED [7] and A-RED [9], and successfully achieves the delicate balance between queue-length minimization and throughput optimization. PURPLE is also able to reduce the number of packet drops due to buffer overflow drastically, creating benefits for both real-time applications for which retransmission delays often are intolerable and reliable transport protocols by reducing end-system buffer-space requirements. The queuing delay is also reduced, which again helps to improve protocol performance.

B. *Organization*

This paper is organized as follows. In Section II we describe the PURPLE algorithm in detail. Section III evaluates PURPLE and compares it with other well-known AQM schemes in several scenarios. Then, Section IV provides the comparison with related work, before the paper is concluded in Section V.

II. PURPLE

Here, we look at how we can use model-based parameter optimization to control the traffic to the desired levels. First, we discuss the model and the validity of its approximations (Sections II-A and II-B). Later, in Section II-C, we describe how to measure the necessary model inputs and in Section II-D we outline some implementation issues.

A. *Static Model*

A simple and frequently used approximation of the steady-state throughput X_n of a single TCP session has been derived by Mathis *et al.* [1] as

$$X_n = \frac{c_n \cdot S_n}{R_n \cdot \sqrt{p_n}} \ ,$$

where the parameters have the following meaning:

- n is the current connection of a bundle of altogether N streams ($1 \leq n \leq N$),
- c_n is a system constant close to 1, which depends on the loss model and the TCP acknowledgment strategy (either acknowledge every packet or delayed ACK),
- S_n is the maximum segment size (often around 1460 bytes),
- R_n is the RTT, and
- p_n is the probability of a "packet loss," or more generally, a packet indicating congestion anywhere between source and destination. (The aggregation of one or more individual indications into "events" by TCP is accounted for by c_n.)

This assumes a *greedy* flow, i.e., a flow that never has to wait for the sending or receiving application.

The parameters c_n and S_n typically are constant for a given TCP connection. Also, R_n is quite stable if we assume queuing delays to be relatively small. Thus the main parameter to control TCP bandwidth is p_n, the congestion indication probability. This leads to one of the key observations behind PURPLE:

If we want to scale the bandwidth X_n of a single TCP flow by a factor γ, we merely need to scale p by $1/\gamma^2$, independent of the other parameters, i.e., there is no need to measure or manipulate c_n, S_n, or R_n.

This can easily be seen by substitution and simplification:

$$\gamma X = \gamma \frac{c_n \cdot S_n}{R_n \cdot \sqrt{p_n}} = \frac{c_n \cdot S_n}{R_n \cdot \sqrt{\frac{1}{\gamma^2} p_n}} \ .$$

As we will see, the loss/marking probability is trivial to measure in an ECN environment. Moreover, it can easily be influenced at a router, with the obvious limitation that no router should unmark (or resurrect) packets that have been marked (or dropped).

Now, let us move from a single connection to a bundle of N simultaneous TCP sessions traversing node k. The total bandwidth in the steady state is then calculated as

$$X = \sum_{n=1}^{N} \frac{c_n \cdot S_n}{R_n \cdot \sqrt{p_n}} \ .$$

Assuming that the c_n and S_n are common among the links, we obtain

$$X = \sum_{n=1}^{N} \frac{c \cdot S}{R_n \cdot \sqrt{p_n}} = c \cdot S \sum_{n=1}^{N} \frac{1}{R_n \cdot \sqrt{p_n}} \ .$$

Instead of treating the N sessions separately, we now simplify them into a bundle of N identical "representative" sessions, whose R is chosen such that p matches the real average drop probability p between source and destination seen:

$$X = N \frac{c \cdot S}{R \cdot \sqrt{p}} \ .$$

To scale this aggregate by γ, we obtain

$$\gamma X = N \frac{c \cdot S}{R \cdot \sqrt{\frac{1}{\gamma^2} p}} + e \ , \tag{1}$$

where e is the error introduced because the formula is not linear in p and because not all contributing flows match the representative session. Under typical traffic loads, the various e_n contributing to the total $e = \sum e_n$ will mostly cancel each other out.

For a congested link, γ is determined as the ratio between the link bandwidth L and the offered load O. (On an uncongested link, a router does not modify p, as it has been set because of a preceding congested link, which should not be further congested.) By knowing γ and the original p, we can thus easily calculate an approximate new p', which will bring the aggregate sender transmission rates much closer to the desired rate. The resulting control loop will quickly converge to the target value. This quick convergence will also manifest itself in the simulation results.

B. Model Dynamics

So far, we have only considered the steady state, assuming a simple total end-to-end probability. We will now split p into three probabilities: $p_{<k}$ is the probability that the packet was marked before our node k, p_k that the packet will be marked at k, and, $p_{>k}$ that it is marked after k. For small probabilities, the approximation $p \approx p_{<k} + p_k + p_{>k}$ holds, which can be turned into an equality if routers only mark previously unmarked packets (this dependency is natural in the packet-loss model, where it is impossible for the same packet to be dropped multiple times). As a result, we obtain

$$X \approx N \frac{c \cdot S}{R \cdot \sqrt{p_{<k} + p_k + p_{>k}}} \ . \tag{2}$$

Traditionally, each router independently decides whether and how much to contribute to the total end-to-end marking probability, based only on its own needs. We largely keep this model, but router k measures $p_{<k}$ and estimates $p_{>k}$ before making its contribution, so that it only applies what needs to be added. The $p_{>k}$ that will be applied to the packets cannot be determined a priori, as packets will be marked in the future. Instead, a simple extrapolation from preceding iteration is used, namely, we assume that nothing has changed in the preceding RTT. Even though this seems too simplistic, we have obtained good results with it, but will keep looking for better estimators.

From the findings of Section II-A, we obtain the control rule as

$$p' \leftarrow \frac{p}{\gamma^2} \ ,$$

where p' is the new marking probability that should be seen in total, γ is again the bandwidth reduction factor, as determined from the current link load. Together with Equation (2), this becomes

$$p'_k \leftarrow \frac{p}{\gamma^2} - p_{<k} - p_{>k} \ , \tag{3}$$

where p'_k is the manipulated variable. p, $p_{>k}$ and γ are obtained in the current time step, but are the result of the control put into place earlier. $p_{>k}$ will only happen later to this packet, therefore, we revert to estimating that the situation has not changed significantly from the preceding time step. Conveniently, the information about the preceding state is delayed an RTT by the network and thus can be observed just in time, as will be discussed further in the next section.

C. Measuring the Variables

Until now, we have assumed that the variables are known. This section describes how their values can be obtained.

$p_{<k}$ is calculated as the ratio of packets that have the ECN CE (Congestion Experienced) code point set in the IP header compared to the total number of ECN-enabled packets.

p is obtained by counting the packets that have the CWR (Congestion Window Reduced) notification flag in the TCP header set, again relative to the total ECN-enabled packets. This indicates the probability of an event having happened, and does not report individual markings. As PURPLE is able to keep the loss probability relatively smooth and low, the number of events closely approximates the number of marks.

For the actual measurement of the ratios, we keep a history of the last 16 CE and CWR events, respectively, together with their arrival point in time, as measured in number of packets. This has the advantage that it allows a quick reaction to congestion but that the rates will also decay reasonably fast when no congestion has been seen.

$p_{>k}$ cannot be directly measured, but is obtained by subtracting p_k and $p_{<k}$, both recorded from the preceding RTT, from the p currently measured, which again reflects events that occurred in the preceding RTT.

The determination of $p_{>k}$ is where R comes into play, which so far we have been able to ignore. However, a reasonably accurate estimate of R is required to determine the delaying of p_k and $p_{<k}$ values between measurement and application. We use a simple mechanism that has proved to be remarkably reliable: For some of the packets marked by router k, it adds an exact-match packet filter that will fire when a packet for the same flow traverses having CWR set.

Limiting the monitoring to just ECN information eliminates the need for detailed knowing of the inner working of the transport protocol, such as sequence number handling and retransmit behavior. The router only needs to touch packets

Algorithm 1 packet_arrival(pkt)

```
if ecn_enabled($pkt$) then
    measure_ce_probability();
    measure_cwr_probability();
    if rand() $\leq p_k$ then
        set_ce($pkt$);
        start_rtt_est($pkt$);
    end if
end if
if Queue full then
    drop($pkt$);
    return;
end if
enqueue($pkt$);
```

Algorithm 2 bg_task()

```
ewma($\overline{q}$);
ewma($L$);
remove_expired_rtt_entries($rtt\_max$);
$p'_k \leftarrow \frac{p}{\gamma^2} - p_{<k} - p_{>k}$ ;
if $\Delta t > --rtt\_cntr$ then
    $\gamma \leftarrow \frac{L}{O}$ ;
    Update $p_{<k}$ from history;
    Update $p_{>k}$ from history;
    Update RTT estimate $R$;
    $p \leftarrow \frac{p}{\gamma^2}$ ;
    $rtt\_cntr \leftarrow$ floor($\frac{R}{\Delta t}$) ;
end if
```

and inspect flags that it would have had to, anyway. Like any RTT measurement in an environment in which packets can get lost and are sometimes retransmitted, this process can also wrongly causally link two events which are only temporally related; i.e., measure the time to a CWR, which was not directly or immediately caused by the CE. To accommodate for such unavoidable measurement errors, we use a window of 32 samples, ignoring the top and bottom 10 percentiles.

D. Implementation

The implementation is outlined in Algorithms 1 and 2, which discuss the per-packet processing and a regularly executed background task, respectively. The `ewma()` functions referenced therein relate to updating the exponentially weighted moving average for the variables mentioned. As was done for the NS-2 implementation of RED, the background task could also be integrated into the per-packet function, and adapting the EWMA time constants accordingly. The use of an EWMA has the advantage of low-pass filtering the values measured. Therefore, any short-term change due to traffic bursts does not result in a significant change of the averaged value. The `floor()` function returns the largest integral value not greater than its argument.

So far we have discussed the macroscopic part of the algorithm in which probabilities are updated in the order of the estimated RTT. As queue occupancy can significantly change during one RTT estimate, we introduce a fine-grained mechanism (microscopic part) which adapts drop probabilities more accurately by taking into account the average queue length. For this, we leave the basic PURPLE algorithm unchanged except that we modify the maximum rate that can be allocated depending on the average queue length. We reduce the rate that is indicated indirectly by PURPLE to the incoming flows in order to gain some headroom to drain the queue.

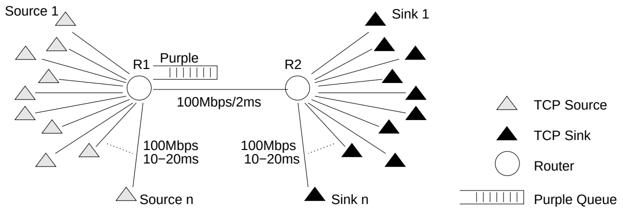


Fig. 1. Simulation topology for the single-bottleneck case.

The maximum allocatable rate X' seen by PURPLE at k is given as

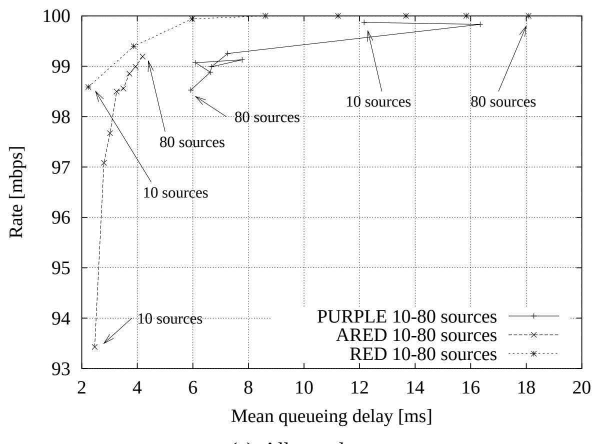
$$X' = \frac{L}{\alpha(1 + \overline{q}/Q)} , \tag{4}$$

where L is the maximum link capacity, $\overline{q}$ the average queue occupancy, Q the total queue capacity, and α can be used to explicitly reduce the average queuing delay, but is usually set to 1. Its influence will be shown later. The time interval for microscopic probability updates is typically handled by a background task (shown in Algorithm 2) in the same frequency as probabilities are updated in network processors [10] and is on the order of milliseconds. In a much more coarse-grained resolution, the set of the PURPLE parameters needed by the macroscopic part is maintained every estimated mean RTT, R.

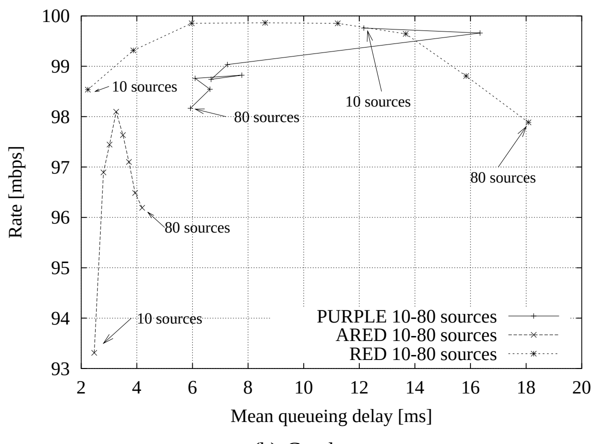
III. EVALUATION

For the evaluation of any Internet protocol or mechanism, the use of real Internet data is of utmost value. Barring a full-fledged deployment in the network backbone, the only practicable alternative is to use Internet traces. Unfortunately, they are not appropriate for analysis of any application that significantly affects the traffic, including AQM and PURPLE. Therefore, we reverted to the use of the network simulator NS-2 [11] for all simulations and compare the results with RED and A-RED.

For our simulations, we first configured queues to achieve high link utilization by setting their capacities in the order of the bandwidth delay product [2], a widely accepted recommendation. Unfortunately, this leads to high queuing delays, and is often not practiced owing to the large amount of buffer space required, which can amount to several hundred megabytes of high-speed memory for fast links. Nevertheless, this simulation helps to understand the robustness of the core part of the PURPLE algorithm. Then we evaluate the full PURPLE algorithm in scenarios with one or more bottleneck links.

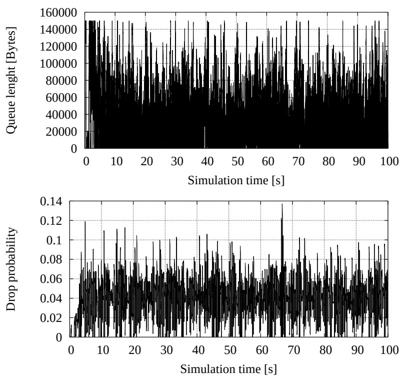


(a) Allocated rate.

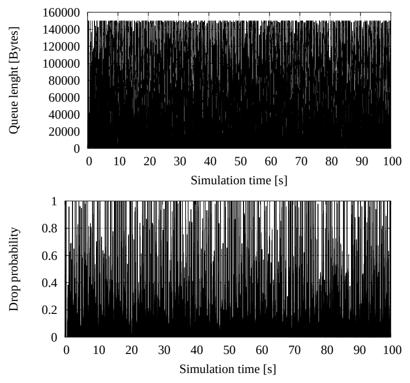


(b) Goodput.

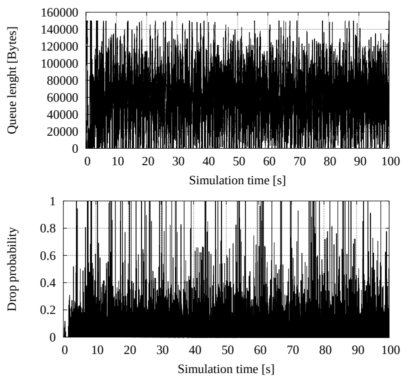
Fig. 2. Rate and goodput comparison of simulations with a set of greedy TCP connections and maximum queue length equal to the bandwidth delay product.



(a) PURPLE.



(b) RED.



(c) A-RED.

Fig. 3. Comparison of average queue length and local drop rate of different AQM schemes width 100 TCP sources. Note the magnified drop probability scale for PURPLE.

A. The Single-Bottleneck Case

A first simulation setting is illustrated in Figure 1. Several greedy and ECN-enabled TCP sources share a single bottleneck link between routers R1 and R2. The number of TCP sources is varied from 10 to 80, and a simulation run lasts 100 s. To compare the PURPLE algorithm with other AQM schemes such as RED and A-RED, the queue at the bottleneck link is configured with a queuing capacity equal to the bandwidth delay product (660 full-length packets). This is particularly important, as the main part of the PURPLE algorithm does not take the average queue length into account. Therefore, the outgoing link bandwidth seen by the PURPLE algorithm in this scenario is not adapted by the average queue length, thus neglecting the micro-managed part of the algorithm completely. The offered load is smoothened using an exponentially weighted moving average with weight $1/32$. The background task is called every millisecond. For RED, the two thresholds th_{min} and th_{max} are 10 and 600 packets, the maximum marking probability is 0.02, and the weight for updating the average queue length w_q is 0.002. A-RED uses the NS-2 default parameters. As we are not interested in the behavior of AQM with varying packet sizes, we set the average packet size of the TCP sources to 1500 Bytes.

Figure 2 shows the allocated link rate at router R1, as well as the TCP goodput. It can be seen that A-RED is not able to profit from the full available queue capacity. Instead, it too aggressively tries to lower queuing delays, at the cost of a significant waste in allocated link rate and overall goodput. Although RED shows excellent rate and goodput results, the queuing delay increases linearly with the number of TCP sources. PURPLE without microscopic drop probability updates typically gets much lower queuing delays than RED, without the A-RED sacrifice in efficiency. In contrast to RED, PURPLE queuing delay tend to be shorter when the number of TCP sources increases indicating higher stability of the algorithm. For PURPLE, as drop probabilities are updated in the order of the estimated RTT the mean queuing delay has a higher variance. The strength of the congestion indication, as manifested in the numbers of marked or dropped packets, is depicted in Figure 4 as a function of the number of

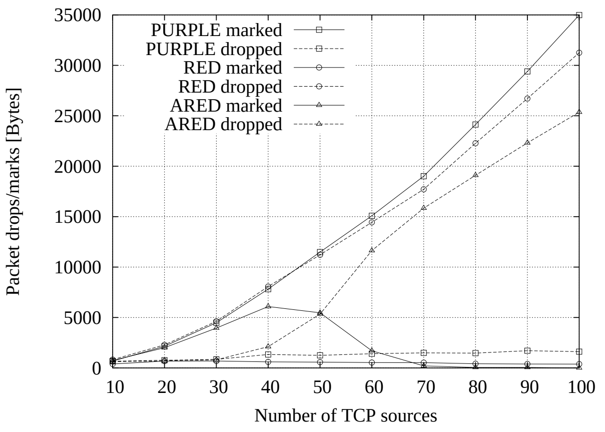


Fig. 4. Mark and drop rates of packets in a simulation with greedy TCP connections and maximum queue length equal to the bandwidth delay product.

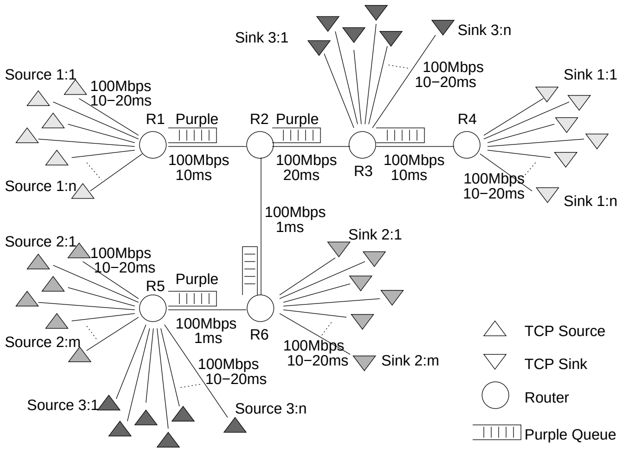


Fig. 6. Simulation topology for the multi-bottleneck case.

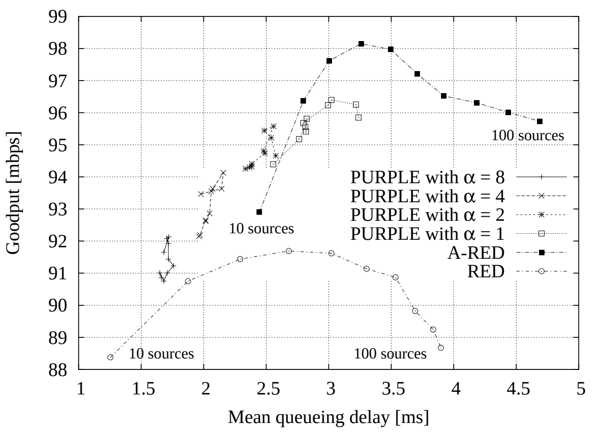


Fig. 5. Goodput and delay comparison of simulations with a set of greedy TCP connections and short queues.

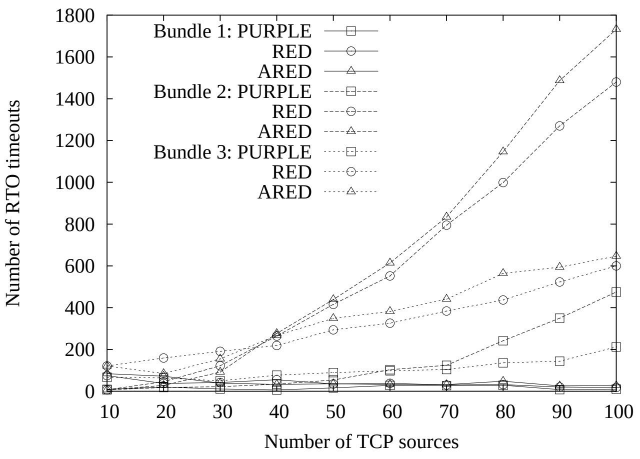


Fig. 7. RTO as a function of the number of TCP sources in bundle 2.

TCP sources. It can be clearly seen that PURPLE, because of its inherent model prediction, significantly reduces expensive packet drops compared with the other two schemes. This results in shorter end-to-end delays thanks to the lower number of packet retransmits.

To reduce unnecessary end-to-end delays, we shortened the queues in a second simulation. At the same time X' is adapted using Equation (4), with $\alpha = 1$. The total queuing capacity is set to 100 packets, and the RED thresholds th_{min} and th_{max} are set to 20 and 60 packets. All other parameters remain unchanged. Queue lengths and drop probability as a function of the simulation time are depicted in Figure 3, where 100 TCP sources are active. As expected, PURPLE is able to maintain lower average queue occupancy and a smooth packet marking rate. Figure 5 shows that PURPLE is in general less sensitive to changes in the number of flows. This shows that PURPLE routers operate very well even under queues much shorter than the generally accepted recommendation by Villamizar [2].

If a network operator has strong needs towards providing either very low delay or very high throughput, the priority parameter α can be tuned, without affecting the other good points of PURPLE, namely smoother traffic and self-tuning. While PURPLE offers this opportunity, we believe that selecting a different value of α will be extremely rare.

B. The Multi-Bottleneck Case

The multi-bottleneck case as depicted in Figure 6 is similar to the previous simulation environment with the exception of the added additional bundles of TCP sources and several nodes that build new congestion points. By varying the link delays we chose to create a scenario in which different PURPLE-enabled routers see an estimated RTT that differs approximately by a factor of two, thus testing the stability of the algorithm. The TCP bundles 1 (sources 1:1 to 1:n) and 3 (sources 3:1 to 3:n) consist of 50 senders ($n = 50$), and bundle 2 varies from 10 to 100 sources. Within each bundle, the RTTs for the TCP sessions are equally distributed in the bounds shown in Figure 6. All other parameters remain unchanged.

Figure 7 shows the number of retransmission timeouts (RTO) events for the three TCP bundles. PURPLE significantly reduces the number of RTO events for all bundles. Furthermore, we found no evidence of TCP synchronization, and goodput and mean queuing delay showed congruent results, found in the single-bottleneck case. Finally, we can state that there are no significant differences between the relative bandwidth allocations of the three bundles.

IV. Related Work

Ever since Erlang started developing what became queuing theory [12], there has been a fundamental tradeoff in single-server systems between long queues or idle servers and the resulting low service rates. With the advent of packet networks, there appeared a possibility to control the arrival rate by notifying the senders. The mechanism used by TCP [13] is to take packet loss as an indication. A less drastic measure was later used in the DECbit feedback mechanism [14], which used a bit in the DECnet protocol header but required symmetric routes. ECN [6] recently brought feedback into the Internet world while avoiding the symmetric-route requirement, which significantly reduced the requirements for retransmits [15].

DECbit already provided a first *early indication* of congestion, i.e., before the queue was full. This was adequate for the small impact (the lack of) congestion had on the transmission speed in DECnet. But in other environments with stronger congestion response, better models were needed. Random Early Drop (RED) [7] started dropping (or, with ECN, marking) already at low queue utilization, but with low probability. The result was that only a fraction of TCP senders would half their transmission window (and indirectly the transmission rate), smoothing the response signal. It was soon realized that RED required tuning according to the expected number of flows and the RTTs seen by these flows, or, in other words, the aggressiveness of the flow aggregate. Many mechanisms were proposed to "automagically" adjust for this [9], [16], [17]. Still, the result is an often bursty queue length and frequent queue overruns.

Similar approaches were used for ATM ABR [18], where a system analogous to TCP with ECN was proposed, but with the capability of signaling several levels of congestion.

Plasser *et al.* [19] studied nonlinear drop probability functions in RED. They argue that traditional linear RED function cannot cope with AQM design criteria such as the avoidance of forced drops and link underutilization. Using the TCP window model and parameters estimated by means of simulations, they propose a nonlinear drop probability function for RED. Especially at low loads, i.e., a low number of TCP sources, these functions maintain significantly higher queue sizes in order to prevent underutilization.

In Dynamic-RED [20], the drop probability is adapted in order to stabilize the queue length close to a user-defined threshold value. Benefits are bounded delays and independence from the number of flows traversing a router.

An orthogonal approach is to disproportionately punish the flows consuming most bandwidth. The first proposal was Flow RED, or FRED [21], which groups the packets currently in the queue into flows. Approximative Longest Queue Drop (ALQD) [22] efficiently identifies some candidates for the longest (virtual) per-flow queues, thereby reducing the computational complexity. BLUE [23] does not want to rely on the queue occupation snapshots. Instead, it aims for a longer-term view and uses therefore packet loss and link under-utilization events Packets are classified into flows. This is done efficiently in Stochastic Fair BLUE by using multiple levels of independent hash functions.

Other rate-based algorithms are GREEN [24] and BAT [5]. GREEN uses an additive-increase and additive-decrease function of the average offered load to evaluate the drop probability. The offered load is estimated from the exponentially smoothened inter-packet delays. BAT is based on per-flow additive increase and multiplicative decrease to achieve fast convergence and low queuing delays.

CLCC [25] introduces a second-tier congestion control mechanism running on top of an AQM scheme such as BAT. As responsive protocols amplify the congestion signal received, non-responsive flows will be able to grab more than their fair share of the bandwidth. This unfairness is counteracted by a control mechanism regulating the drop preference bias between the two classes of traffic.

Other applications of TCP modeling include work on TCP-friendly flow control [26], multicast congestion [27], and control of misbehaving flows, including possible denial-of-service attacks [28]. It has also been proposed to modify bottleneck access routers so that they fairly share their link bandwidth among the flows originating locally, without the TCP-typical bias toward short RTTs [29].

V. Conclusions

PURPLE provides smooth packet marking rates and queuing delay without any tuning of parameters because of its online optimized, autonomous behavior that relies on online model-based predictions It is also able to avoid tail-drop losses almost completely while providing an excellent balance between goodput, throughput, and average delay. This is achieved using minimal effort and state information thanks to the introduction of three new mechanisms, namely, end-to-end congestion analysis, monitoring of ECN information, and use of the TCP model equation. Using simulations, we have shown that PURPLE behaves very well under a variety of circumstances. We believe that moving away from a pure-local view to a wider angle would help improve other AQM methods as well.

The significantly reduced packet loss by using PURPLE routers results in a large improvement of predictability of TCP traffic delays. This is significant and could even rehabilitate TCP for the use in many time-critical implementations which currently are implemented on top of UDP.

In the future, we will investigate further ways to cleverly use ECN information. We also plan to perform analyses in more realistic settings. Unfortunately, the public availability of packet traces is of no use in AQM evaluation, as such traces obviously will not adapt according to the AQM signals. We are planning to participate in larger simulations [30] or real-world testbeds. We also plan to evaluate PURPLE in the environment of the Alpha-Beta traffic mix proposed by Wang et al. [31].

REFERENCES

[1] M. Mathis, J. Semke, and J. Mahdavi. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), 1997.

[2] Curtis Villamizar and Cheng Song. High performance TCP in ANSNET. *Computer Communications Review*, 24(5):45–60, October 1995.

[3] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. In *Proceedings of 7th International Workshop on Quality of Service (IWQoS '99), London*, pages 260–262, June 1999.

[4] Mikkel Christiansen, Kevin Jeffay, David Ott, and F. Donelson Smith. Tuning RED for web traffi c. In *Proceedings of SIGCOMM 2000*, pages 139–150, August 2000.

[5] E. Bowen, C. Jeffries, L. Kencl, A. Kind, and R. Pletka. Bandwidth allocation for non-responsive fbws with active queue management. In *Proceedings of Int. Zurich Seminar on Broadband Communications, IZS 2002*, February 2002.

[6] K. K. Ramakrishnan, Sally Floyd, and David L. Black. The addition of explicit congestion notifi cation (ECN) to IP. RFC 3168, Internet Engineering Task Force, September 2001.

[7] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[8] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of SIGCOMM*, pages 303–314, 1998.

[9] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An algorithm for increasing the robustness of RED. Technical report, ICSI, August 2001.

[10] Robert Haas, Clark Jeffries, Lukas Kencl, Andreas Kind, Bernard Metzler, Roman Pletka, Marcel Waldvogel, Laurent Frelèchoux, and Patrick Droz. Creating advanced functions on network processors: Experience and perspectives. *IEEE Network*, To appear, 2003.

[11] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala. Improving simulation for network research. Technical Report 99-702b, University of Southern California, March 1999.

[12] Agner Krarup Erlang. The theory of probabilities and telephone conversations. *Nyt Tidsskrift for Matematik B*, 20:33, 1909.

[13] Jon Postel. Transmission control protocol. RFC 793, Internet Engineering Task Force, September 1981.

[14] K. K. Ramakrishnan and Raj Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Transactions on Computer Systems*, 8(2):158–181, 1990.

[15] J. Hadi Salim and U. Ahmed. Performance evaluation of explicit congestion notifi cation (ecn) in ip networks. RFC 2884, Internet Engineering Task Force, July 2000.

[16] K. Chandrayana, B. Sikdar, and S. Kalyanaraman. Scalable confi guration of RED queue parameters. In *Proceedings of HPSR*, pages 185–189, May 2001.

[17] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. A self-confi guring RED gateway. In *Proceedings of the 17th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99)*, pages 1320–1328, March 1999.

[18] Y. Zhao, S. Li, and S. Sigarto. A linear dynamic model for design of stable explicit-rate abr control schemes. Technical Report 96-0606, ATM Forum, April 1996.

[19] Erich Plasser, Thomas Ziegler, and Peter Reichl. On the non-linearity of the RED drop function. In *Proceedings of International Conference on Computer Communication (ICCC)*, August 2002.

[20] J. Aweya, M. Ouellette, D. Y. Montuno, and A. Chapman. A control theoretic approach to active queue management. *Computer Networks*, 36:203–235, 2001.

[21] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of SIGCOMM '97*, pages 127–137, Cannes, France, September 1997.

[22] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury. Effi cient active queue management for Internet routers. In *Proceedings of Interop Engineers Conferece, Las Vegas, NV*, May 1998.

[23] W. Feng, D. Kandlur, D. Saha, and Kang G. Shin. BLUE: A new class of active queue management algorithms. Technical Report CSE-TR-387-99, University of Michigan, April 1999.

[24] Bartek Wydrowski and Moshe Zukerman. GREEN: An active queue management algorithm for a self managed internet. In *Proceedings of ICC 2002*, volume 4, pages 2368–2372, April 2002.

[25] Roman Pletka, Andreas Kind, Marcel Waldvogel, and Soenke Mannal. Active queue management for fair bandwidth allocation of mixed responsive and non-responsive traffi c using a closed-loop congestion control scheme. Research report, IBM, February 2003.

[26] Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer. Equation-based congestion control for unicast applications. Technical Report TR-00-003, International Computer Science Institute, March 2000.

[27] Sherlia Shi and Marcel Waldvogel. A rate-based end-to-end multicast congestion control protocol. In *Proceedings of ISCC 2000*, pages 678–686, Antibes, France, July 2000.

[28] Ratul Manajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network. *ACM Computer Communication Review*, 32(3), July 2002.

[29] Mohamed A. El-Gendy and Kang G. Shin. Equation-based packet marking for assured forwarding services. In *Proceedings of IEEE Infocom*, pages 845–854, June 2002.

[30] Andreas Kind and Bernard Metzler. Rate-based active queue management with token buckets. In *Proceedings of 6th IEEE International Conference on High Speed Networks and Multimedia Communications (HSNMC'03)*, July 2003.

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