TADA: An Active Measurement Tool for Automatic Detection of AQM

February 1, 2016

Abstract

The problem of overbuffering in today's Internet (termed as bufferbloat) has recently drawn a great amount of attention from the research community. This has led to the development of various active queue management (AQM) schemes. The last years have seen a lot of effort to show the benefits of AQMs over simple tail-drop queuing and to encourage deployment. Yet it is still unknown to what extent AQMs are deployed in the Internet. In this paper, we present an end-to-end active measurement method to detect AQMs on the path bottleneck. We have developed an active measurement tool, named *TADA*, and evaluated our measurement methodology on a controlled experimental testbed. Experimental results show that the proposed approach provides the basis to identify whether an AQM is deployed on the bottleneck.

1 Introduction

The classical approach for handling packets in a router is to use tail-drop FIFO queueing. It establishes a single queue for each outgoing link, and forwards packets on that link in the order of arrival. Packets are dropped, from the tail of the queue, only when the queue is full. To maximise the link utilisation, and minimise loss, operators typically configure large packet buffers in their routers. However, research has shown that oversized buffers have resulted in large standing queues, and consequently increased end-to-end latency, a phenomenon that has come to be known as "bufferbloat" [9]. Bufferbloat occurs when very large buffers in the network, in combination with simple queueing schemes, create excessive delays due to TCP flows probing for bandwidth. Bufferbloat has been measured in different parts of the Internet, ranging from ADSL to cable modem users [7, 16, 20].

As awareness of the topic of Bufferbloat has risen, so too has the interest in methods to resolve it. Active queue management (AQM) schemes appear to be the most promising approach because significant network-wide benefits can be derived by implementing AQMs on the access network elements (e.g., broadband modems) [22]. AQMs prevent or minimize delay by managing the queue of packets intelligently [19, 8, 18]. They are able to outperform the classic tail-drop solution in many ways, for example by avoiding excessive queue buildup and preventing flow synchronization.

Although AQMs have been extensively studied [11, 15, 17, 22] and have proved to outperform the classic tail-drop solution, it is only in the last few years, through the efforts in the bufferbloat project and others [6, 5], that it has started to get a foothold in deployed edge routers [1, 4]. However, we do not know to which degree the efforts towards promoting AQMs have convinced the ISPs and equipment manufacturers to deploy AQMs instead of large tail-drop queues. Answering this question, as the first step, requires a method that can detect the presence of AQM-enabled routers in a network.

This detection problem can as well be viewed as an instance of a new class of network tomography [21] problems. Instead of estimating internal link delays, losses, or the network topology, in this class of tomography problems the objective is to identify the type of forwarding modules that a packet flow goes through. Knowing this may help engineers develop more efficient computer networks and increase quality of service. As an example, some transport protocols will perform poorly over certain AQMs [10, 17]. Knowing about the AQM deployed on the bottleneck may provide Operating System and application designers with information that can help them make better decisions for their services. In addition, knowing whether a path contains an AQM could be beneficial for future network performance measurement studies and development of new active measurement tools. For example, some existing active measurement tools [13, 14, 12] assume that ISPs mostly use tail-drop queues. Knowing about the presence of an AQM in the path invalidates this assumptions and may inform on the accuracy of the results.

In this paper, we present an active measurement tool, called *TADA* (Tool

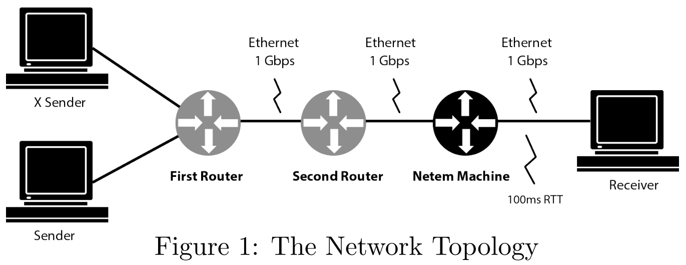


Figure 1: The Network Topology

for Automatic Detection of AQMs), that can detect if the bottleneck router on a particular communication path uses AQM. Our detection technique is based on analyzing the patterns of `queue delays` and `packet losses`. We use two statistics, namely Pairwise Comparison Test and Pairwise Difference Test, defined in [12] to maintain accuracy even in the presence of background traffic, which can cause dramatic fluctuations in the measurements. We have evaluated the tool using an experimental testbed in a controlled laboratory environment. As per the proposed approach, the tool is able to detect the presence of an AQM on the bottleneck router.

The rest of the paper is organized as follows: First we explain the key idea of our measurement approach in Section 2. Section 3 describes *TADA* in detail. Then, we show experimental results in Section 4, which verify the tool accuracy. Finally, Section 5 concludes the paper and discusses potential future work.

2 Design

In this section, we first present the basic idea for differentiating between AQM and tail-drop. We then proceed by explaining the detection method in detail in Section 2.2 and Section 2.3.

2.1 Basic Idea

Consider a path from a `Sender` to a `Receiver` as illustrated in Figure 1. Suppose that the capacity of the path bottleneck is C, and that `Sender` transmits an isochronous stream at a constant bit rate $R_s > C$ to `Receiver` for a duration of t seconds. The stream consists of N maximum-segment sized (MSS) packets. We detect if the bottleneck in the `Sender-to-Receiver` path uses AQM. We do this detection at the `Receiver`.

When the stream rate R_s is larger than the bottleneck available bandwidth C, the packets begin to build up a queue at the bottleneck. If the bottleneck employs tail-drop as its queue management scheme, and t is long enough, the queue will exceed the available buffer at some point. A tail-drop only drops packets when there is no space left in its buffer. On the other hand, active queue management schemes do not wait until their buffer is saturated. AQM schemes start dropping packets out of their buffer as soon as they detect the

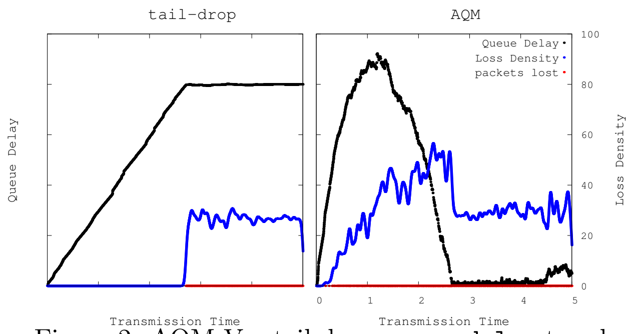


Figure 2: AQM Vs. tail-drop `queue delay` trend

queue is growing too large [19, 8, 18].

In our work the main idea for differentiating between AQMs and tail-drop is to detect if packets are dropped while the queue is still growing. To do so, we record `packet loss` information, and use the trend in packets' `queue delay` to provide an insight into the queue growth.

Suppose that `Receiver` receives a stream of K packets ($0 \leq K \leq N$). $N-K$ packets have been dropped by the bottleneck router. `Receiver` computes the `queue delay` for each received packet based on the packets transmission time t_i, and arrival time a_i. To do so, `Receiver` calculates the one-way delay (OWD) for all delivered packets as $D_i = a_i - t_i$ first; the `queue delays` of packet i would be $Q_i = D_i - D_m$, where D_m is the minimum OWD[1]. Since $R_s > C$, as long as the queue is building up, the packets' `queue delays` have an increasing trend. Meaning that packet i will wait in the queue for a longer time duration than packet $i-1$.

When the bottleneck's queue is full, the packets' `queue delay` reaches its maximum, and will stop increasing. In this case, if the queue management is tail-drop then new packets can get into the queue at the same rate the packets leave the queue, causing a constant `queue delay` trend. Otherwise, if the queue management is an AQM, the `queue` delay trend will eventually become decreasing[2].

To analyse this idea for AQM detection, we have tried to simulate the continuous constant bit rate traffic between sender and receiver via a bottleneck. And we have tried to analyse the trend for `queue delay` and `loss pattern`. Simulation results are presented in Figs. 3-

Recall that, in the case of tail-drop, no packets are lost before the queue is full. This does not hold if we have an AQM. Therefore, we can distinguish between tail-drop and AQM by investigating the first part of the `queue delay` curve until its maximum. We refer to this part, which is monotonically increasing, as the *first increasing part* of the `queue delay` trend. Combining the *first*

[1]Since we are only interested in the relative `queue delays` not the absolute ones, the presence of a clock offset does not influence these measurements.

[2]In the case of ARED, it eventually becomes constant. However, as opposed to tail-drop, some packets are lost before the `queue delay` trend becomes constant.

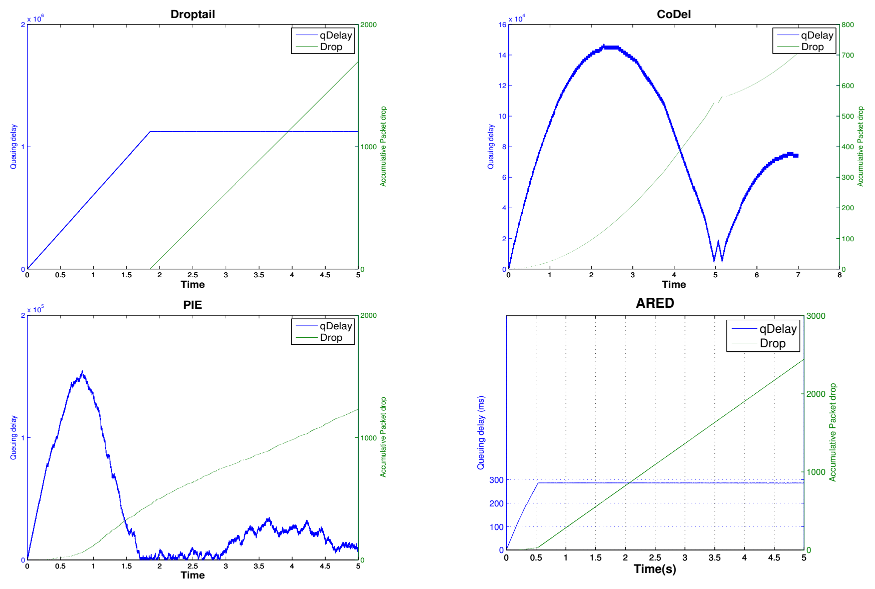


Figure 3: Queueing delay and loss pattern results for AQM simulation

increasing part with `loss occurrences` in the same period, `Receiver` can infer whether any loss happened while the queue was growing and detect whether an AQM is used.

Besides `queue delay` and `loss occurances`, `Receiver` uses another metric to be able to differentiate between AQM and tail-drop in various scenarios. Using a Gaussian kernel density estimator [2], `Receiver` calculates `loss density` for every packet i as L_i (Equation 1). Each sequence number i is considered as a data point $(1 \leq i \leq K)$. $|i-j|$ is the distance from data point i to the packet loss with sequence number j.

$$L_i = \sum_{Over\ j} \frac{1}{\sqrt{2\pi}} e^{-\frac{|i-j|}{2}} \tag{1}$$

`Loss density` could be interpreted as the bottleneck drop scheme loss distribution. `Loss density` information helps `Receiver` detecting tail-drop from AQM in the presence of high load background traffic. It is also useful in distinguishing between different AQMs.

Figure 2 depicts sample measurements of `queue delay` and `packet loss` for tail-drop and AQM[3]. A tail-drop shows zero `loss density` at first, a significant increase in loss occurrences when there is no more space in its buffer, and stays almost constant after. While, AQMs `loss density` has a correlated relationship with its `queue delay`. Applying the method mentioned in previous paragraphs

[3]In Figures 2-8 we intentionally removed the numbers on the y-axis, since we are only interested in the trend not the exact values.

and by just looking at the graphs, we can easily detect that the graph on the left corresponds to a tail-drop and the right one is an AQM. However, to automate the detection process in a real environment, we need to use mathematical and statistical analysis techniques to reliably identify the `queue delay` trend, and find its *first increasing part*, and maximum.

In the following, we first explain how the `queue delay` information is used to find the *first increasing part*. Then, we explain how we add `packet loss` and `loss density` information to detect the presence or absence of AQM.

2.2 Finding the First Increasing Part

As mentioned in the previous section, the key idea of our approach is to find the *first increasing part* of the `queue delays`' trend and check if any packets were lost in that period. In general, and particularly in realistic environments over Internet where the traffic pattern is complex, finding the desired increasing part of the `queue delay` trend is not easy. The complexity is due to the fluctuation in the packets' `queue delay`. Jain et al [12] have proposed a reliable algorithm to smooth a fluctuated curve. They employed two complementary statistics, called *Pairwise Comparison Test (PCT)* and *Pairwise Difference Test (PDT)*. We use the same method to identify the `queue delay` trend and find its *first increasing part*.

2.2.1 PCT and PDT

Let $\mathcal{Q} = \langle q_1, q_2, ..., q_K \rangle$ be a sequence of measurements regarding queuing delays corresponding to the sequence of packet transmission times $\mathcal{T} = \langle t_1, t_2, ..., t_K \rangle$ (i.e., q_i is the `queue delay` experienced by the packet that was sent at t_i). We first partition $\mathcal{Q}$ into $\Gamma = round(\sqrt{K})$ groups of measurements. Then, for each group $j \in \{1, ..., \Gamma\}$, we compute its median $\hat{q}_j$, and define $\hat{\mathcal{Q}}$ as the sequence $\langle \hat{q}_1, ..., \hat{q}_\Gamma \rangle$. Note that the sequence of medians $\hat{\mathcal{Q}}$ is more robust than $\mathcal{Q}$ to outliers and errors.

The PCT metric of $\hat{\mathcal{Q}}$ is calculated as follows:

$$PCT(\hat{\mathcal{Q}}, n) = \frac{\sum\limits_{j=2}^{n} I(\hat{q}_j > \hat{q}_{j-1})}{n-1} \tag{2}$$

Where $2 \leq n \leq \Gamma$ and $I(X)$ is one if X holds, and zero otherwise. PCT measures the fraction of consecutive measurement pairs that are increasing, and so $0 \leq$PCT ≤ 1. The PDT of $\hat{\mathcal{Q}}$ is:

$$PDT(\hat{\mathcal{Q}}, n) = \frac{\hat{q}_n - \hat{q}_1}{\sum\limits_{j=2}^{n} |\hat{q}_j - \hat{q}_{j-1}|} \tag{3}$$

Where $2 \leq n \leq \Gamma$. PDT evaluates the strength of the variation in the measurements. Note that $-1 \leq$ PDT ≤ 1. If there is a strong increasing trend, PCT and

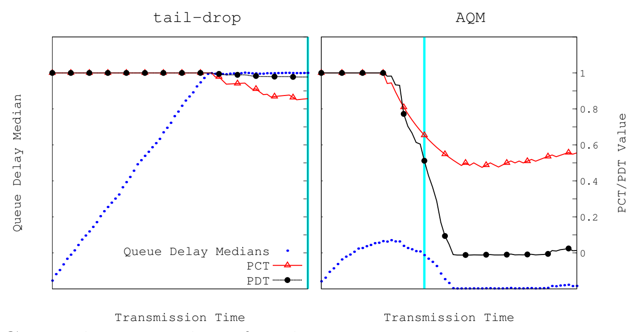


Figure 4: PCT and PDT values for the `queue delay` measurements in Figure 2

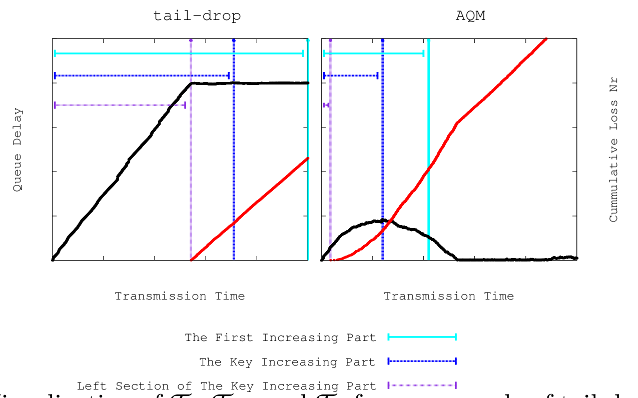


Figure 5: Visualization of $\mathcal{T}_I$, $\mathcal{T}_M$, and $\mathcal{T}_L$ for an example of tail-drop (left), and an example of AQM (right).

PDT approach to one. The authors of [12] identify a set of values as increasing if either $PCT > 0.66$ and $PDT > 0.45$, or $PDT > 0.55$ and $PDT > 0.54$. In our work we use the same thresholds.

2.2.2 Algorithm for finding the increasing part

In our approach, we use the same statistics (i.e., PCT and PDT) to find the *first increasing part* of the `queue delays` $\mathcal{Q}$. In other words, the goal is to find the largest sequence of `queue delays` that have an increasing trend, starting from q_1. Algorithm 1 presents the pseudo code we used to find the increasing part of the `queue delays`. The algorithm iterates over $2 \leq g \leq \Gamma$, and computes $PCT(\hat{\mathcal{Q}}, g)$ and $PDT(\hat{\mathcal{Q}}, g)$; checks them against the desired range specified above and updates the *index* value. The return value of the algorithm is the largest g for which $PCT(\hat{\mathcal{Q}}, g)$ and $PDT(\hat{\mathcal{Q}}, g)$ are in the desired range.

If neither of the conditions in lines 7 and 8 of Algorithm 1 are ever satisfied throughout the algorithm, resulting in $index = 0$, we cannot use the collected `queue delay` information for detection. In this case, the queue management scheme is *unknown*. Otherwise, $\mathcal{Q}_I = \langle q_1, ..., q_g \rangle$ forms the *first increasing part*.

We use $\mathcal{T}_I$, a sub-sequence of $\mathcal{T}$ starting at t_1 and ending at t_g, to denote the time sequence corresponding to $\mathcal{Q}_I$.

Algorithm 1

```
1: $g := 2$
2: $index := 0$
3: procedure SeparateIncPart($\hat{\mathcal{Q}}$)
4:     while $g \leq \Gamma$ do
5:         $pct := PCT(\hat{\mathcal{Q}}, g)$
6:         $pdt := PDT(\hat{\mathcal{Q}}, g)$
7:         if ($pct > 0.66$ and $pdt > 0.45$) or
8:             ($pdt > 0.55$ and $pct > 0.54$) then
9:             $index := g$
       return $index$
```

Figure 4 shows PCT and PDT values calculated in each iteration of Algorithm 1. The blue points are the medians of `queue delays` ($\hat{\mathcal{Q}}$) for the examples in Figure 2. In each graph, the cyan line shows the largest *index* returned by Algorithm 1.

2.3 Detection

If Algorithm 1 return $index = 0$, then we do not have useful data to make a conclusion. Consequently, we decide that the queue is unknown. Otherwise, after finding the *first increasing part* of the `queue delays` trend, we use the recorded `loss occurrence` and `loss density` information to detect the bottleneck queue management. Following we first describe how we separate tail-drop from AQM. Then we proceed by describing our method for differentiating between ARED and CoDel/PIE. Finally, we investigate the possibility of distinguishing between PIE and CoDel.

2.3.1 AQM or tail-drop

In the first step towards detecting AQM from tail-drop, we check if any packets were lost in the *first increasing part* (i.e., $\mathcal{T}_I$). If there are no packets lost at the increasing part of the `queue delays`, our job is completed. The bottleneck queue management is tail-drop. This happens when the bottleneck queue is very large. `Sender` transmits a stream for a duration of t seconds. The packets fill up the bottleneck queue, but they do not saturate it. As a result, while `queue delays` has an increasing trend, there is not any packet lost in that period. Therefore, `Receiver` can infer the bottleneck queue is tail-drop.

However, if there are some packets lost, there is still a chance that the queue management scheme is tail-drop. This might happen because the increasing part we find is an approximation of the actual increasing part of the trend. Consequently, it might include a non-increasing part of the `queue delay` trend.

In Figure 5 we used cyan lines to mark the *first increasing parts*, which in the case of both graphs include a portion of the non-increasing part of the trend.

If there are some packets lost during $\mathcal{T}_I$, we first exclude the non-increasing part by finding the maximum value of `queue delays` (q_M) in the increasing part selected by Algorithm 1. Let t_M be the packet transmission time corresponding to q_M. We refer to the time interval between t_1 and t_M as the *key increasing period*, and denote it with $\mathcal{T}_M$. In Figure 5 the *key increasing period* and $\mathcal{T}_M$ are marked using blue lines.

Again, if there are not any packets lost during $\mathcal{T}_M$, we conclude that the path's bottleneck is using tail-drop. Otherwise, we need to further inspect the combination of the `queue delays` and the `loss occurrence` information during the *key increasing period* to be able to make a decision.

We first divide the *key increasing period* $\mathcal{T}_M$ into two sections $\mathcal{T}_L$ and $\mathcal{T}_R$. The left part ($\mathcal{T}_L$) starts from t_1 to the point where the first `packet loss` occurred, denoted as t_l. The right part $\mathcal{T}_R$ contains the rest of the sub-sequence $\mathcal{T}_M$ (i.e., starts from t_{l+1} and ends at t_M). $\mathcal{T}_L$ and t_l are marked using purple lines in Figure 5.

After finding $\mathcal{T}_L$ and $\mathcal{T}_R$, we compute the slope of the line connecting (t_1, q_1) to (t_l, q_l), and the slope of the line connecting (t_{l+1}, q_{l+1}) to (t_M, q_M). If the left part slope is more than τ times larger than the right part slope, we infer that the bottleneck queue management is tail-drop. Otherwise, the bottleneck queue management scheme is most probably AQM. However, in the presence of heavy background traffic, a tail-drop bottleneck might produce a very similar `queue delay` trend to an AQM. Therefore, to increase the accuracy, we would look at the packets' `loss density` during ($\mathcal{T}_R$). If the bottleneck queue scheme is AQM, the packets' `queue delay` and `loss density` during ($\mathcal{T}_R$) would be highly correlated, while the correlation between `queue delay` and `loss density` would be less than or approximately equal to zero for a tail-drop, depending on the situation.

Table 1 summarizes all the notations we have introduced in this section.

2.3.2 ARED Vs. CoDel/PIE

The next step in the detection procedure is to differentiate between ARED and CoDel/PIE. First, let us look at the sample measurements of ARED and PIE in Figure 6. Recall that we removed the non-increasing part of the *first increasing part* of the `queue delays` by finding maximum queue delay in the *first increasing part* and called it the *key increasing part*. The difference between PIE(CoDel) and ARED lies in the non-increasing part of the *first increasing part* we found in Section 2.2. We marked the non-increasing part by cyan lines in Figure 6. Each subfigure includes a small graph containing median `queue delays` for the non-increasing part. We can see that the `queue delay` medians for ARED gets constant after maximum queue delay because ARED tries to keep `queue delay` between minimum and maximum thresholds. However, PIE(CoDel) would behave differently. PIE(CoDel) `queue delay` has a decreasing trend after max `queue delay`. Consequently, to detect ARED from

Table 1: Notations and assumptions

Parameter	Description
C	Capacity of the link
N	Total number of transmitted packets
K	Number of successfully received packets
R_s	Transmission rate
t	Transmission duration
ϵ	constant used to update transmission rate
l	`packet loss` threshold
$\mathcal{Q}$	Set representing `queue delay` measurements
$\mathcal{T}$	Set for sequence of transmission times

PIE(CoDel), we would look at PCT and PDT values for the non-increasing period.

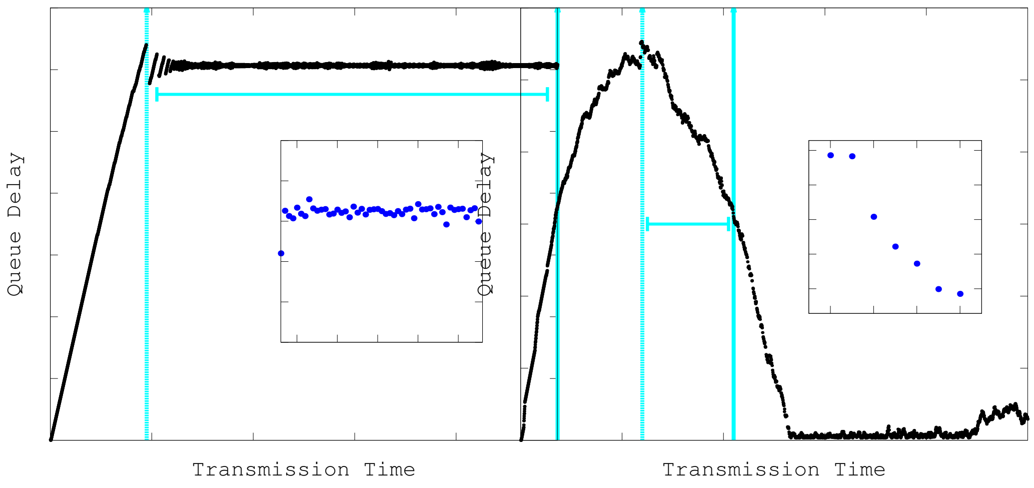
If the bottleneck queue is ARED, PCT would have an independent value around 0.5 and PDT would be around zero. Otherwise, PCT would be approximately zero and PDT would have a high negative value around -0.8. This implies that the specific period has an decreasing trend which leads us to the conclusion that the bottleneck queue management scheme is either CoDel or PIE.

2.3.3 CoDel Vs. PIE

3 TADA

We have developed a measurement tool, called *TADA*, that actively detects if an AQM scheme is deployed on the middlebox. The tool is composed of a `Sender` process running at the sender machine and a `Receiver` process running at the receiver machine. The tool uses UDP for the constant bit rate (probing) stream and TCP for a control channel between the endpoints.

TADA works in an iterative manner. In each iteration r, `Sender` transmits a constant bit rate stream with rate $R_s(r)$ to `Receiver` for a fixed amount of time t. `Receiver` collects `queue delay` and `packet loss` information for the whole period. It then analyzes this information to detect the presence of AQM based on the idea described in section 2. We use the path capacity C as the starting transmission rate $R_s(0)$, and increase it by a factor of $(1+\epsilon)$ in each iteration. In Section 3.1, we describe an approach for estimating the path capacity C. By using C as the starting transmission rate and increasing it in each iteration, we maximize the chance that the packets sent by the sender are queued somewhere in the `Sender-to-Receiver` path. Note that if queuing does not happen during the transmission phase, the receiver will not be able to record any interesting



(a) ARED Queue Delay (b) PIE Queue Delay

Figure 6: PIE and ARED sample measurement - The small figures shows median queue delays for the non-key insreasing prt

data for detection. In addition, if the transmission rate is too large, it may congest a wrong bottleneck (i.e., a router before the actual bottleneck of the path). To avoid this, we use a very small ϵ for updating the transmission rate.

In the following, we first describe our approach for estimating the path capacity. Then we explain the detection loop in which we use the detection method presented in section 2.

3.1 Capacity Estimation

To estimate the path capacity we send N number of MSS-sized (size S) back-to-back packets to the `Receiver`. Suppose that $P \leq N$ packets are successfully received. The `Receiver` estimates the path capacity as: $C = \frac{(P-1)S}{\sigma}$ (where σ is calculated as the difference between received timestamps of the last and first packets). In our implementation, we use $N = 500$ packets.

3.2 Detection Loop

In each iteration r, we apply the detection method described in Section 2 on the collected `queue delay` and `packet loss` information to check if an AQM is used. If the method detects tail-drop, the algorithm terminates reporting the absence of an AQM. Otherwise, the algorithm has either detected an AQM on the bottleneck queue or the queue type is unknown. In both cases, a new iteration

$r+1$ starts with a higher transmission rate calculated as $R_s(r+1) = (1+\epsilon)R_s(r)$. To increase the transmission rate, we keep the packet size unchanged (MSS size) and only decrease the inter-transmission time (ITT) between packets. We repeat this iterative process until a high enough `packet loss` rate (i.e, larger than $l\%$) is reached or tail-drop is detected. In the end, *TADA* terminates by reporting that the queue management is either tail-drop, AQM, or unknown.

4 Evaluation

In this section, we evaluate the accuracy of the proposed measurement approach in a controlled testbed.

4.1 Experimental setup

The controlled testbed consists of two sender machines (i.e., `Sender` and `XSender`) and one receiver machine (i.e., `Receiver`) connected through a three-hop path as shown in Figure 1. The first two intermediate boxes between the senders and the receiver are Linux routers: `First Router` and `Second Router`. We used another machine to emulate network delay using *Netem* [3] (`Netem Machine`). The delay is $50ms$ in each direction. All the machines run Debian Linux on a 4.0.0 kernel. `XSender` was used to produce background traffic. In our experiments we examined several scenarios:

- **Single bottleneck**: In this case, `First Router` has $1Gbps$ bandwidth and `Second Router` has $10Mbps$ bandwidth, making `Second Router` the bottleneck of the path.

- **Serial bottlenecks**: Here, we use two different values for `First Router` bandwidth: $\{15Mbps, 100Mbps\}$, and kept the `Second Router` bandwidth $10Mbps$.

We repeat each experiment 50 times. Each experiment had a certain level of background traffic load: (1) no load, where we have no background traffic, (2) low load, where we only have short flows (generated using a TCP traffic generator[4]) as background traffic, (3) medium load, where both short flows and one greedy TCP flow are present, and (4) high load, where there are short flows and three greedy TCP flows.

We investigated *TADA*'s accuracy for tail-drop against three different parameterless AQM variants, namely CoDel, PIE, and ARED for which we have used the LARTC `tc` tool for emulation. In our experiments, we used $t = 5s$, $\epsilon = 0.1$, and $l = 20\%$ (as defined earlier in Table 1).

4.2 Results

4.2.1 Without background traffic

In the first phase of our evaluation, we have used `TADA` to detect the presence of AQM for the scenario where there is no other flow, results are illustrated in

[4]The traffic generator opens a TCP connection every $50ms$, requesting a Pareto distribution with download size $\alpha = 0.9$ having minimum size of $1KB$. Every request by `Receiver` opens a new TCP connection, closed by the `XSender` after sending the data.

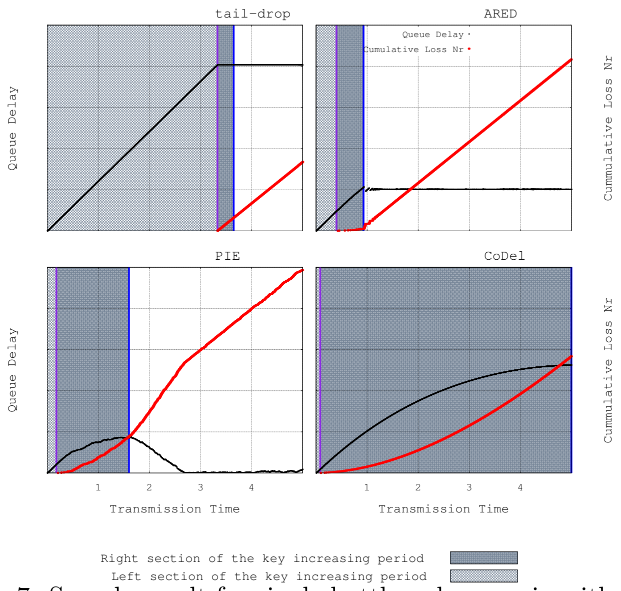


Figure 7: Sample result for single bottleneck scenario with no load

Figure 7. Each graph in Figure 7 corresponds to a different queue management scheme; namely tail-drop, ARED, PIE, and CoDel. The *left* and *right* parts of the *key increasing part* (defined in Section 2) are marked using two different background patterns and separated by vertical lines. `TADA` is able to detect bottleneck queue management schemes accurately in 100% of the test runs for this scenario. However, there are varying background traffic patterns on the Internet. We therefore want to investigate how the tool performs with different background traffic loads, as discussed below.

4.2.2 With background traffic

In the second phase, we have performed a series of experiments using different background traffic loads as described earlier. Figure 8 shows a sample result for a single bottleneck path and in the presence of high load background traffic. The same visual aesthetics as used in Figure 7 are applied to show the results for different queue management schemes.

Table 2 reports the accuracy of *TADA* for each scenario. As shown in this table, *TADA* is able to correctly detect the presence of an AQM in most cases, even with a loaded bottleneck. Detection failure are due to various reasons, such as high load background, or presence of multiple lossy routers (e.g., serial bottlenecks with very similar bottleneck capacities). In addition, depending on the access technology and path conditions, the network may randomly drop some packets. This can affect the accuracy of *TADA*.

Table 2: Accuracy of detection for the scenarios where the bottleneck is subjected to background traffic.

Scenario	**Load**	**Accuracy**
Single bottleneck	Low	100%
	Medium	100%
	High	97%
Serial bottleneck (100Mbps)	Low	100%
	Medium	98%%
	High	89%
Serial bottleneck (15Mbps)	Low	100%
	Medium	70%
	High	50%

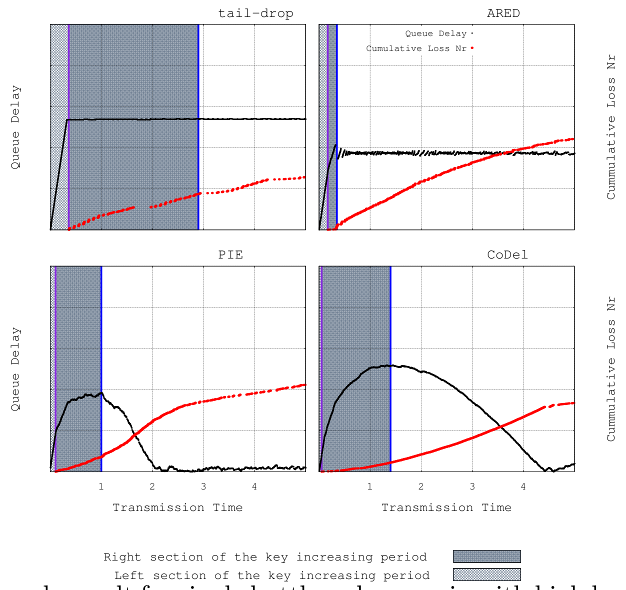


Figure 8: Sample result for single bottleneck scenario with high load background

5 Conclusion and future work

We have presented a novel approach that can detect whether a bottleneck router uses an AQM as its queue management scheme. We have evaluated our proposed approach by developing an active measurement tool and evaluated it in a controlled testbed setup. The testbed results show that the tool is able to accurately detect the presence of an AQM on the bottleneck router when there is no background traffic. The detection is also accurate in the presence of low background traffic like short flows or one greedy flow. However, if the background traffic is high load, like more than 3 greedy flows, the accuracy goes down. Nevertheless, we can estimate the load of the background traffic and discard the detection process if the background traffic is too high.

For future work, we intend to investigate the possibility of separating different AQMs from each other. In addition, we plan to explore multiple concurrent flows to identify scheduling (*flow queueing*) effects in parallel queues.

References

[1] Cerowrt router firmware for fighting bufferbloat. `http://www.bufferbloat.net/projects/cerowrt`. Accessed: 2015-09-13.

[2] Kernel Density Estimators.

[3] Netem - Linux network emulator. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`. Accessed: 2015-03-25.

[4] Openwrt router software. `https://openwrt.org/`. Accessed: 2015-09-17.

[5] RITE Project. `http://www.riteproject.eu/`. Accessed: 2014-06-15.

[6] The Bufferbloat projects. `http://www.bufferbloat.net/`. Accessed: 2015-06-15.

[7] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing Residential Broadband Networks. In *Proc. of IMC'07*, 2007.

[8] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management. Technical report, 2001.

[9] J. Gettys. Bufferbloat: Dark buffers in the Internet. *IEEE Internet Computing*, 15(3), 2011.

[10] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. Taht. Fighting the bufferbloat: On the coexistence of aqm and low priority congestion control. In *INFOCOM, 2013 Proceedings IEEE*, pages 3291–3296, April 2013.

[11] E. Grigorescu, C. Kulatunga, and G. Fairhurst. Evaluation of the impact of packet drops due to AQM over capacity limited paths. In *Proc. of ICNP'13*, 2013.

[12] M. Jain and C. Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *Proc. of PAM'02*, 2002.

[13] P. Kanuparthy and C. Dovrolis. Diffprobe: Detecting ISP service discrimination. In *INFOCOM*. IEEE, 2010.

[14] P. Kanuparthy and C. Dovrolis. Shaperprobe: End-to-end detection of ISP traffic shaping using active methods. In *IMC'11*. ACM, 2011.

[15] N. Khademi, D. Ros, and M. Welzl. The new AQM kids on the block: An experimental evaluation of CoDel and PIE. In *Proc. of INFOCOM'14 WKSHPS*, 2014.

[16] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the Edge Network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, 2010.

[17] N. Kuhn, E. Lochin, and O. Mehani. Revisiting old friends: Is CoDel really achieving what RED cannot? In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, CSWS '14, 2014.

[18] K. Nichols and V. Jacobson. Controlling queue delay. *Queue*, 10(5), May 2012.

[19] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A lightweight control scheme to address the bufferbloat problem. In *HPSR*. IEEE, 2013.

[20] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband Internet performance: A view from the gateway. In *Proc. of SIGCOMM'11*, 2011.

[21] Y. Vardi. Network tomography: Estimating source-destination traffic intensities from link data. *Journal of the American Statistical Association*, 91(433):365–377, Mar. 1996.

[22] G. White and D. Rice. Active queue management in DOCSIS 3.1 networks. *Communications Magazine, IEEE*, 53(3):126–132, 2015.