FLOW-VALVE: EMBEDDING A SAFETY-VALVE IN RED

Kenjiro Cho
Sony Computer Science Laboratories, Inc.

Abstract

In this paper, we present the flow-valve, a safety-valve mechanism for RED to protect the network from misbehaving or overpumping flows and to promote end-to-end congestion control. The flow-valve can be regarded as an implementation of the concept known as a "RED penalty-box" but our focus is to protect router resources in times of congestion. The flow-valve detects a traffic increase that goes beyond the control range of RED, and protect the local resources by forcing overpumping flows to back off. The flow-valve provides an incentive for end-to-end congestion control to keep the packet drop rate low under moderate congestion, and to conservatively back off under heavy congestion. Our simulation results demonstrate that the flow-valve can effectively protect the network from misbehaving flows and, at the same time, isolate undesirable behavior of conformant TCP. The flow-valve also has been successfully implemented onto FreeBSD.

1 Introduction

The flow-valve is a safety-valve for RED in order to protect a network from greedy flows and to promote end-to-end congestion control. The idea of the flow-valve is suggested in [FJ93, FF99] and has been known as a penalty-box in the research community. Although RED substantially improves the performance of a network of cooperating TCP flows, RED is known to be vulnerable to greedy flows. Protecting the RED mechanism by regulating misbehaving flows, at the same time, can provide an incentive for better end-to-end congestion control models, which in turn leads to a more robust and more scalable global Internet.

Floyd *et al.* in [FF99] argue on the need for end-to-end congestion control, and further, on the need for mechanisms in the network to detect and restrict unresponsive or high-bandwidth best-effort flows in times of congestion. The idea is to provide an incentive in support of end-to-end congestion control for best-effort traffic. Several approaches and mechanisms are discussed but it is not clear how to implement such a model in an efficient manner.

Our goal is to create a rough approximation of the theoretical model and make a simple yet effective prototype implementation available in order to solicit real world experiences along the direction proposed in [FF99]. The important issue is, as pointed out in [FF99], to make such systems available and gain field experience rather than pursuing a precise mechanism.

2 Background

The flow-valve is an enhancement to RED and a simple implementation of the concept known as "penalty-box". We first review the RED queue management, the penalty-box model, analytical TCP studies, and then, show the impact of TCP timeouts to the traditional penalty-box model.

2.1 RED (Random Early Detection)

Random Early Detection (RED) [FJ93] is an active queue management mechanism that drops (or marks) incoming packets with a probability corresponding to the average queue length. To calculate the drop probability, the average queue length avg is compared to two thresholds, a minimum threshold min_{th} and a maximum threshold max_{th}. As avg varies from min_{th} to max_{th}, the drop probability linearly increases from 0 to the maximum drop probability max_p.

RED is proved to keep the average queue length short while allowing occasional bursts of packets, to avoid the synchronization of flows, to be roughly fair, and to improve the utilization of the network. However, RED itself does not have a protection mechanism against uncooperative flows, and thus, an unadaptive flow can force RED to fall back to the Drop Tail behavior.

Fair Random Early Drop (FRED) [LM97] tries to improve RED by adding per-active-flow accounting. FRED employs a flavor of per-flow buffer management by maintaining state for flows having packets in the queue. FRED is also able to regulate misbehaving flows to some extent. Our approach is similar to FRED in adding a limited number of per-flow states but differs in that FRED does not provide an incentive for end-to-end congestion control.

2.2 Penalty-box

The RED mechanism from its introduction in [FJ93] has the idea of identifying misbehaving flows. The idea is further

developed in [FF99, FFT98]. We briefly review the arguments in [FF99].

It is a great threat to the current Internet that a growing amount of traffic does not use congestion control. The lack of end-to-end congestion control causes the unfairness that, in times of congestion, responsive flows reduce their sending rate while unresponsive flows use up the available bandwidth. It also leads to the danger of various types of congestion collapse in which the performance of the network is drastically deteriorated by uncontrolled packets.

The problems of unresponsive flows can be solved either by per-flow scheduling, by end-to-end congestion control, or by pricing. Among the three approaches, however, only end-to-end congestion control provides the right incentive for cooperation and sharing that are essential to the operation of the Internet.

To promote the use of end-to-end congestion control, routers need mechanisms that detect and restrict the bandwidth of uncooperative flows by means of preferential packet scheduling or packet dropping.

Three tests are proposed to identify flows to regulate. (1) the TCP-friendly test is to see if the packet arrival rate of a flow is no more than a conformant TCP session. It provides the upper bound of a TCP throughput by the packet drop rate, the minimum round-trip time and the maximum packet size. (2) the unresponsiveness test is to see if the arrival rate decreases appropriately in response to an increased packet drop rate. The important observation is that, if the packet drop rate of a flow increases by a factor of x, the packet arrival rate should decrease by a factor of at least $\sqrt{x}$. (3) the disproportionate-bandwidth test is to see if the flow is using a significantly larger share of the bandwidth than other flows.

These basic ideas are not specific to RED but the model described in [FF99] uses the RED queue management. Floyd *et al.*, further in [FFT98], investigate a way to use the RED packet drop history to estimate the packet arrival rates of flows.

The idea of identifying misbehaving flows and regulating them has come to be called a "penalty-box" in the research community, though the word "penalty-box" is not used in [FF99]. The concept of a penalty-box is simple but there are difficulties to implement a penalty-box. One is to efficiently obtain statistical information of a flow in a network that is highly dynamic by its nature. In addition, routers have limited information about flows. Another is to formally define "misbehaving" or "TCP-friendly" since TCP or other sophisticated transport mechanisms are a collection of complicated algorithms. Yet another difficulty is how to regulate a misbehaving flow.

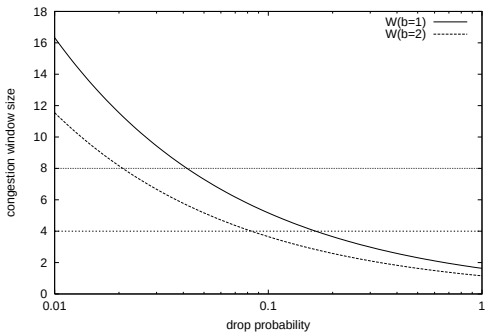


Figure 1: packet drop rate and maximum congestion window size in the steady state TCP model

2.3 TCP Behavior at a High Packet Drop Rate

The behavior of TCP is well-studied in recent research [MSMO97, FF99, PFTK98, Floyd91, FJ92, LU97]. Especially, the behavior of TCP when RED is actively dropping packets is of our interest.

When packet drop is rare (much less than 1%), TCP can sustain its sending rate by Fast-Retransmit/Fast-Recovery. As the packet drop rate increases, retransmissions become driven by timeouts. The steady-state model of TCP provides the upper bound of the congestion window W with a non-bursty average packet drop rate p [MSMO97, FF99, PFTK98].

$$W = \sqrt{\frac{8}{3bp}} \tag{1}$$

Here, b is the number of packets that are acknowledged by a received ACK, and is typically 2 since most TCP implementations employ ack-every-other-packet policies.

As p increases, W becomes smaller. Figure 1 shows how W changes with $b = 1$ and $b = 2$. When the congestion window becomes less than 4 segments, TCP is no longer able to recover from a single packet loss since Fast-Retransmit needs at least 3 duplicate ACKs to get triggered. When packet drops are randomly phased, W needs to be at least 8 because the congestion window oscillates between $W/2$ and W. Thus, it can be concluded that, when p is more than 0.02, the throughput of normal TCP is severely damaged.

Mathis *et al.* [MSMO97] also investigate the transition from the congestion avoidance behavior at moderate p to the timeout driven behavior at larger p with several TCP implementations by the *ns* simulator. The transition is observed to start at $p < 0.01$ for normal TCP. Kumar [Kumar98] uses a stochastic model to study the throughput of various versions of TCP over lossy wireless links. His analytical model based on a Markov renewal-reward process also shows that the transition starts at $p < 0.01$.

Table 1: Reno TCP behavior with different packet drop rate

drop rate	packets/sec	rexmits/sec	timeouts/sec
0.0025	180.90	0.40 (0.2%)	0.01 (0.0%)
0.0050	173.06	0.69 (0.4%)	0.03 (0.0%)
0.0075	153.46	1.15 (0.7%)	0.09 (0.1%)
0.010	140.92	1.55 (1.1%)	0.07 (0.1%)
0.025	89.83	2.37 (2.6%)	0.34 (0.4%)
0.050	55.72	2.75 (4.9%)	0.68 (1.2%)
0.075	38.28	2.87 (7.4%)	1.04 (2.7%)
0.10	26.43	2.86 (10.8%)	1.27 (4.8%)
0.25	4.83	1.26 (26.1%)	0.82 (17.0%)
0.50	0.43	0.18 (41.9%)	0.15 (34.9%)

From these studies, it is clear that, when the packet drop rate is higher than 0.01, normal TCP in the current Internet is not able to sustain its sending rate. The behavior of TCP becomes driven by timeouts; TCP needs to wait for a retransmission timer to expire, and then, sets the congestion window size to 1 and performs slow-start. At a retransmission, the timeout value is doubled for exponential backoff.

When a timeout occurs, TCP stalls for a substantial period depending on the round-trip time and variance. The coarse timers used in the TCP implementations also have a great impact to the timeout duration [AP99]. For example, BSD based systems have a timer granularity of 500ms and its minimum timeout duration is 2 ticks; the minimum timeout is 750ms on average. The penalty of a timeout is by orders of magnitude larger than that of Fast-Recovery.

Timeouts result in a wide range of variations in TCP behavior, which is problematic for conformance tests in the penalty-box model. A statistical test is applicable only over sufficiently large samples. The number of required samples becomes much larger by timeouts but the packet arrival rate becomes much lower by timeouts.

We investigate the behavior of TCP by the *ns* simulator [MF95]. Table 1 and Figure 2 show the throughput of Reno TCP with varying packet drop rate. (NewReno TCP performs slightly better than Reno TCP but the difference is marginal.) The round-trip latency of the link is 56ms and the bottleneck link bandwidth is 1.5Mbps. The detailed settings are described in Section 4. Note that the default TCP timer resolution is 100ms in the *ns* simulator so that the minimum timeout is 150ms on average. The penalty of timeouts is smaller than most TCP implementations.

In Table 1, a TCP session is observed for 100 seconds with different drop probabilities. The numbers show the unit-time occurrence of packet transmission (including retransmission), retransmission and timeouts for each packet drop probability. The number of packet transmission falls down as the packet drop probability increases, implying that it takes a considerable amount of time to obtain enough samples.

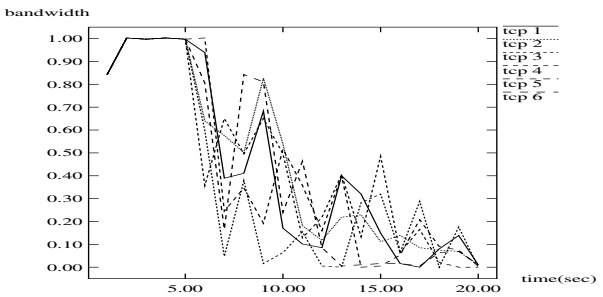


Figure 2: behavior of single TCP sessions

Figure 2 illustrates variations among different sessions. The throughput of a TCP session is observed with an increasing drop rate p; $p = 0$ for the first 5 seconds, $p = 0.025$ for the second 5 seconds, $p = 0.05$ for the third 5 seconds, and $p = 0.1$ for the last 5 seconds. The graph plots 6 runs of a single TCP session and the throughput is measured at 1 second intervals. Although the throughputs, as a whole, follow the rate reduction rule, large variations are observed among different TCP sessions. It indicates that, for this time scale, statistical tests are not applicable to a TCP session because of a limited number of samples and a large variation in the behavior.

In this paper, we focus on the behavior of TCP but the same rules should be applied to other protocols for the best-effort Internet. Because TCP is overwhelmingly dominant in the current Internet and will continue to be so in the foreseeable future, any transport mechanism introduced to the Internet should not be more aggressive than TCP at any level so that existing TCP will not starve.

3 Flow-valve

The flow-valve is a mechanism that detects a traffic increase that goes beyond the control range of RED, and cuts off the flow causing the overload to protect responsive flows and router resources. RED falls back to the simple Drop Tail behavior when the average queue length exceeds max_{th}, which means the traffic is getting out of control. It is likely that the traffic increase is caused by a flow not cooperating with others, and blocking the uncooperative flow will bring the queue length back in the proper range.

The flow-valve borrows many ideas from [FF99] but differs in that our focus is to engineer RED to work in the proper range even in the face of misbehaving flows. Our engineering challenge is to design a mechanism that works with a small number of samples or a transient condition, and approaches the theoretical model as the sample number increases or as the flow state becomes steady.

The flow-valve presents a model similar to the penalty-

box but in a different light. That is, our model is a "safety-valve" instead of a "penalty-box", designed for easy protection and management of network resources at routers by employing two simple policies.

The first policy is to detect an "overpumping" flow instead of a "misbehaving" flow. "Overpumping" means that the sender is transmitting packets more than it should be as perceived by a router along the path. To identify misbehavior, a router needs evidence of misbehavior, which is the source of the difficulties in the penalty-box model. However, identifying overpumping is a decision local to the router experiencing congestion, and the router does not need to prove misbehavior on the sender side. A simple test, *the overpumping test*, is performed locally at each router to detect overpumping flows.

The second policy is to simply block a flow judged as overpumping at a router until the sender backs off exponentially. A simple test, *the backoff test*, is performed to observe exponential backoff.

One might think that blocking a flow is too simplistic and too damaging. However, it turns out that the penalty of a short blocking period is not so different from a passive measurement approach for most TCP implementations. Given the high packet drop rate of an overpumping flow, the probability of timeouts is already quite high without forcing it. As we have shown in section 2.3, the throughput of a TCP session is already severely damaged when the packet drop rate becomes this level. We will study the impact of the blocking scheme in section 4.3.

In addition, a simple blocking scheme has several advantages over measuring rate reduction. The first advantage is quick reaction to traffic surge. Because our goal is to protect the network, the mechanism should respond without delay to offensive flows and a certain-to-work mechanism is needed to protect routers. A statistical approach is not suitable to this end.

The second advantage is that it is more effective in dissolving congestion. If the traffic load reaches the level that RED is no longer able to control, the congestion should be quickly dissolved.

The third advantage is bounded penalty. If a stochastic penalty were used, an unfortunate flow could be punished repeatedly. An exponential backoff mechanism is deterministic and can be observed in a fixed time period.

The fourth advantage is the emphasis on the backoff behavior. We believe that both timeouts and exponential backoff are essential to best effort traffic in order to avoid congestion collapse, though the importance of timeouts and exponential backoff has not been addressed much in previous research. There are many proposals for TCP to avoid timeouts and improve performance but those approaches do not help reduce the packet drop rate at a busy bottleneck link. Traditional TCP implementations are conservative in backing off at a high packet drop rate, which lowers the risk of congestion collapse. In some sense, more aggressive TCP implementations exemplify the lack of an incentive to reduce the packet drop rate.

The fifth advantage is the fairness of the penalty for conservative implementations. The flow-valve, by blocking the arriving packets and observing the backoff behavior, does not allow an aggressive retransmission policy to perform better than others. The only way to not be judged as overpumping is to keep the flow's packet drop rate low. Once judged as overpumping, the penalty is equal for all.

The flow-valve provides an incentive for congestion control; a user needs to keep the packet drop rate low under moderate congestion and to conservatively back off under heavy congestion.

In spite of the differences in our approach, the resulting mechanism is not so different from the original penalty-box model. The flow-valve can be regarded as an implementation of the penalty-box model.

3.1 Flow-list

The flow-valve needs to keep per-flow states but only a small number of per-flow states are required. Because the flow-valve detects flows that use more than a certain fraction of the link bandwidth, the number of such flows is limited at one time. For example, there are at most 10 flows that use more than 10% of the link bandwidth. A few dozens of per-flow states managed by a simple cache scheme should be enough to catch overpumping flows. Although the flow-valve could fail to detect minor misbehaving flows, it is acceptable since our goal is not to detect every misbehaving flow but to protect the RED mechanism.

In addition, we assume an overpumping flow manifests itself during a transient traffic increase. During such a period, a significant fraction of dropped packets will belong to the overpumping flow. Thus, the required flow-list size is much less than that for steady-state traffic.

The flow-valve keeps track of three parameters for each flow. (1) p_{avg} is the average packet drop rate of the flow. (2) f_{avg} is the flow's average fraction of the aggregate packet arrivals. (3) t_{last} is the timestamp of the last packet drop from the flow. p_{avg} and f_{avg} are used for the overpumping test and measured using exponentially-weighted moving average (EWMA). t_{last} is used for the backoff test.

3.2 Overpumping Test

The overpumping test is used to detect a flow causing overload. When the traffic is under the control of RED, the average queue length stays between min_{th} and max_{th} and

the packet drop rate stays less than max_p. RED stochastically drops packets according to the traffic load and responsive flows control their sending rates in response to the packet loss. In such a dynamic traffic environment, a flow that adapts better to the network condition is likely to have a lower packet drop rate than a flow that adapts less because adaptive flows back off during congestion. RED is designed to keep the packet drop rate under max_p with cooperative TCP flows. Thus, if a flow's drop rate p_{avg} exceeds max_p, it is an indication that the flow is not adapting well or not adapting at all. Therefore, we set the packet drop rate threshold p_{th} to max_p and detects flows with $p_{avg} > p_{th}$. It is clear that, if we simply block flows whose packet drop rate is more than max_p, the RED packet drop rate never exceeds max_p.

However, we need to exclude flows using less than their share of the bandwidth because those flows suffer packet drops caused by other flows. Therefore, f_{avg} should be checked as a supplementary test. A fixed threshold could be used to check f_{avg} but a simple function of p_{avg} is used to calculate a reasonable threshold for the packet arrival rate. This function $f_{th}(p)$ is developed later based on a rough approximation of the TCP-friendly model. For now, suppose there is a reasonable function $f_{th}(p)$. Then, we can judge a flow to be overpumping when

$$(p_{avg} > p_{th}) \;\; AND \;\; (f_{avg} > f_{th}(p_{avg})) \tag{2}$$

Note that the overpumping test is to detect the flow causing the overload, and thus, misbehaving flows could stay undetected since the flow-valve is triggered only when p_{avg} exceeds p_{th}. On the other hand, p_{avg} of a responsive flow could exceeds p_{th}. There are a number of possible reasons for that. For example, a large window size and a large RTT could lead to a burst of packet drops. It is also legitimate for TCP to sustain the sending rate at a high packet drop rate when packet drops are fairly uniformly distributed.

In some sense, the flow-valve compensates for the unfairness caused by the TCP mechanism. It is well known that a smaller round-trip time has a clear advantage over a larger one, and thus, a TCP flow with a small round-trip time could be very greedy. Another example is the first slow-start of a TCP session. In the first slow-start, TCP does not know the point (called $ssthresh$) to start the congestion avoidance algorithm, and often results in a burst of packet drops. The flow-valve works as a protection mechanism against such behavior of conformant TCP.

3.3 Backoff Test

The backoff test is used to free a blocked flow. A blocked flow is freed when the retransmission interval becomes more than a backoff threshold d_{th}. Since all the arriving packets are dropped, the sender will continue to double the retransmission interval until the retransmission interval reaches d_{th}. It can be easily detected by a drop timestamp t_{last}.

To observe the exponential growth in the retransmission interval, d_{th} should be an exponentially distributed random value. In practice, a fixed threshold can be used along with a coarse timestamp since rounding errors effectively provide randomization and it is not necessary to check retransmission intervals more than a few seconds.

3.4 Packet Arrival Rate Function

For the overpumping test, we need a simple function of the flow's packet drop rate to estimate a reasonable bandwidth share in order to judge overpumping. We derive a rough approximation of the TCP throughput model using the knowledge of the queue state. However, our goal is to derive a simple approximation that can be used to judge overpumping and the function is not necessarily a precise model of TCP. Even if we had a precise model, it would not work with a small number of samples or transient conditions.

There are a number of analytical studies on the TCP throughput [MSMO97, Kumar98, PFTK98]. The model in [PFTK98] is suitable for large p since it assumes retransmission timeouts, exponential backoff and large RTTs. When the throughput of TCP is not limited by the maximum window size, the throughput in packets $B(p)$ is approximated by:

$$B(p) \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2)} \tag{3}$$

The approximation assumes p is small but it is shown that the model fits well to the measurements over a wide range of p.

Equation (3) can be further simplified by eliminating variables other than p. TCP calculates the retransmission timeout value T_0 (also known as RTO) [Jacobson88, Jacobson90] by:

$$RTO = srtt + 4 \cdot rttvar \tag{4}$$

Thus, we can assume $T_0 > RTT$. We also assume $b = 2$ to reflect ack-every-other-packet policies. Then, the throughput in packets per RTT $B_{rtt}(p)$ is given by:

$$B_{rtt}(p) = B(p) \cdot RTT < \frac{1}{\sqrt{\frac{4p}{3}} + min(1, 3\sqrt{\frac{6p}{8}})p(1+32p^2)} \tag{5}$$

$B_{rtt}(p)$ shows how many packets a TCP session can transmit per RTT. $B_{rtt}(p)$ overestimates the throughput because we ignore $rttvar$ in (4) and the original model does not assume coarse timer granularity. $B_{rtt}(p)$ in Figure 3 reveals that TCP can send only two packets per RTT when

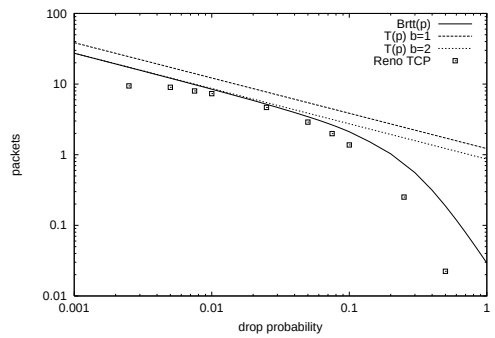


Figure 3: TCP throughput in packets per RTT

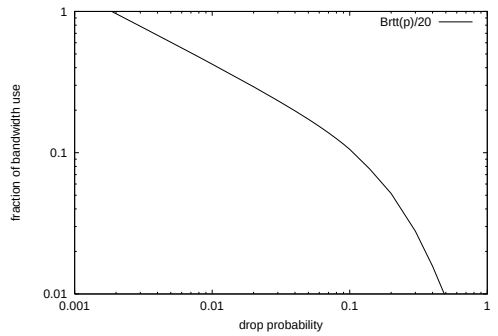


Figure 4: estimated bandwidth share as a function of packet drop rate

$p = 0.1$. We also plot $T(p)$ derived from the original TCP-friendly test in [FF99] with $b = 1$ and $b = 2$. Since $T(p)$ does not assume timeouts, $T(p)$ differs from $Brtt(p)$ when $p > 0.01$. The square marks in the graph show our simulation results in Table 1. The numbers are simply calculated using the link latency in the simulation as RTT. The plot confirms that Equation (5) provides a good estimation for a wide range of p.

Next, we use $Brtt(p)$ to estimate the flow's share of the bandwidth using the knowledge of the queue state. We know that, if p is large, this router is a bottleneck for the flow. The flow's RTT includes the queueing delay at this node and the queueing delay at this node must be a significant fraction of the end-to-end delay. Let avg be the average queue length. The queueing delay at this node can be approximated by the packet service time for avg packets. A single flow is supposed to have less than $B_{rtt}(p)$ packets in the queue. Then, the flow's bandwidth share f becomes

$$f < \frac{B_{rtt}(p)}{(avg + \alpha)} \tag{6}$$

Equation (6) can be used to estimate the flow's share of the bandwidth. In Equation (6), α is an additional factor of the latency. In practice, it is reasonable to set several packets to α if we take into account the buffers inside the network interface cards. If the propagation delay of the attached link is known to be large, it can be added too. There are other factors that possibly contribute to RTT; queueing delay at other routers, (store-and-forward) forwarding delay at the routers, or congestion of the reverse path.

For the overpumping test, Equation (6) can be further simplified. When the overpumping test needs to check f_{avg}, the flow's packet drop rate p_{avg} already exceeds p_{th} and we need to check f_{avg} to exclude a flow using less than its share. If a flow is not using more than its share but the flow's packet drop rate is more than p_{th}, avg is supposed to be more than max_{th} since p_{th} is set to max_p. Therefore, f should be

$$f < \frac{B_{rtt}(p)}{(max_{th} + \alpha)} \tag{7}$$

Equation (7) approximates the reasonable share for a TCP-friendly flow as a simple function of p. For an efficient implementation, the function $f_{th}(p)$ could be implemented in a lookup table, or it could be a fixed threshold for $p = p_{th}$. Figure 4 shows Equation (7) with $max_{th} = 15$ and $\alpha = 5$, and it is in the range appropriate for the overpumping test; $f_{th}(0.05) = 0.17$, $f_{th}(0.1) = 0.1$, $f_{th}(0.2) = 0.05$ and $f_{th}(0.5) = 0.01$.

The function $f_{th}(p)$ is derived using several assumptions. The assumptions may not hold for some environments but the model is better than a heuristic fixed value since it can be easily verified. However, the function is supplementary to the overpumping test and errors in the estimation do not have a significant impact.

In this model, we assume all packets have the equal service time, implying that all packets are equal in size. The RED mechanism can be implemented in either the packet mode or the byte mode [FJ93] and the above model corresponds to the packet mode. In the byte mode, the average queue length counts the queue size in bytes so that it is straightforward to extended our model for the byte mode to take small packets into consideration.

3.5 Scalability

The flow-list maintains a fixed small number of per-flow states regardless of the number of active flows. Thus, the flow-valve can detect flows having a significant fraction of dropped packets but may not be able to detect minor unresponsive flows. In a backbone network, a single flow uses a much smaller fraction of the link bandwidth so that it becomes harder to detect it in a backbone network. On the other hand, this scheme is scalable in the sense that the required state does not grow as the number of flows increases.

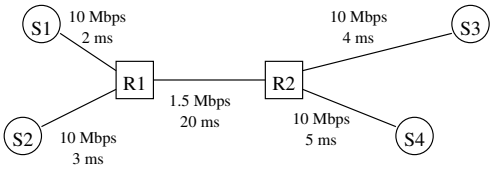


Figure 5: simulation network

Therefore, the flow-valve in a backbone network provides a protection mechanism but does not try to detect minor unresponsive flows.

3.6 Algorithm

Because of the limited space, we do not include the details of the flow-valve algorithm but they are described in the original technical report of this paper [Cho99].

4 Simulation Results

This section presents the simulation results to illustrate the behavior of the flow-valve. Two scenarios are used in the *ns* simulator (version 2.1b3) [MF95] with a simple topology in Figure 5. S1 and S2 are traffic sources and S3 and S4 are traffic sinks. R1 is a bottleneck router and RED and the flow-valve are enabled at R1. The RED parameters are configured with $(min_{th} = 5, max_{th} = 10, max_p = 0.1)$. The queue size limit is 25. The flow-valve parameters are configured with $(p_{th} = 0.1, d_{th} = 1)$. Reno TCP is used for the simulation.

In Figure 6 through Figure 8, graph (a) shows the sequence number of packets observed at R1. Sequence number n of Flow i is plotted at $((n \ mod \ 90)/100 + i)$ so that each flow corresponds to each main row and the sequence number wraps around every 90 packets. A packet is marked in gray when it dequeued, and a dropped packet is marked as 'X' in black. Graph (b) shows the bandwidth use of each flow measured at 0.25 second intervals in Figure 6 and 7, at 0.5 second intervals in Figure 8. The bandwidth use is normalized to the link bandwidth.

4.1 TCP Behavior in Section 2.3

For Table 1 and Figure 2 in Section 2.3, a ftp type flow is used from S1 to S3 with a 20 segment window. Packets are dropped at R1 with a specified probability.

4.2 Test 1

Test 1 is a 25-second-long sequence that illustrates two typical scenarios; one is that an unresponsive flow is detected by the flow-valve during a traffic surge, and the other is that a high-bandwidth flow is quickly throttled by the flow-valve.

The following 4 flows, two ftp flows and two constant bit rate (CBR) flows, are used.

Flow 1 ftp from S1 to S3 (20 segment window size)

Flow 2 ftp from S2 to S3 (5 segment window size)

Flow 3 CBR from S2 to S4 (800Kbps, 1000B/packet)

Flow 4 CBR from S1 to S4 (1.6Mbps, 1000B/packet)

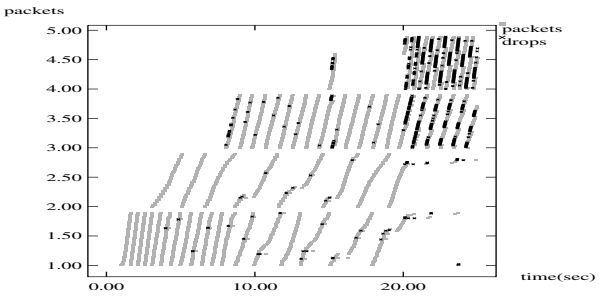
In the beginning of the scenario, two TCP flows, Flow 1 and Flow 2, share the bandwidth. The throughput of Flow 2 is limited by the small window size. At time 8, a CBR flow, Flow 3, starts up and the other two TCP flows reduce their sending rates. At time 15, another CBR flow, Flow 4 is invoked for 0.3 seconds to emulate a traffic surge. At time 20, Flow 4 starts up again, and it lasts longer this time.

Figure 6 shows how the original RED works in Test 1. After Flow 4 starts for the second time, the average queue length reaches max_{th} and RED falls back to the Drop Tail behavior. The TCP flows back off and the two CBR flows use up the available bandwidth.

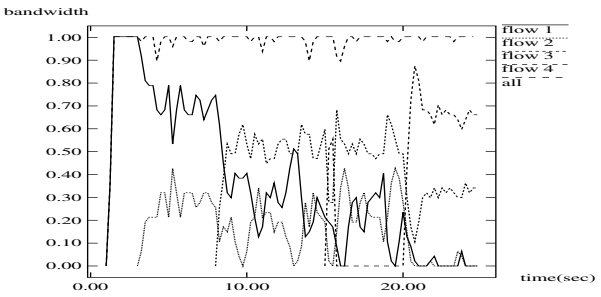
Figure 7 shows the effect of the flow-valve in Test 1. When Flow 3 starts up, the estimated packet drop rate of Flow 3 jumps up, but then, gradually decreases as the other TCP flows reduce their sending rates. The flow-valve does not detect Flow 3 as overpumping at this point since the packet drop rate of Flow 3 is less than the threshold p_{th}. The RED average queue length stays between min_{th} and max_{th}, and the packet drop rate of the TCP flows stay at 2%–3%.

When Flow 4 is invoked to emulate a traffic surge, the sudden increase of the arriving packets causes the packet drop rate of Flow 3 to exceed p_{th}. Flow 3 is judged as overpumping and blocked from then on. This illustrates a typical behavior of an unresponsive flow in the flow-valve that an unresponsive flow manifests itself at a traffic increase. Since Flow 3 is CBR and never backs off, it is never freed. After Flow 3 is blocked, the two TCPs are able to use the full bandwidth again.

When Flow 4 starts again at time 20, Flow 4 is quickly detected as overpumping and blocked by the flow-valve. This illustrates the flow-valve's quick reaction to misbehaving flows. The average queue length is kept below max_{th} in the face of misbehaving flows.

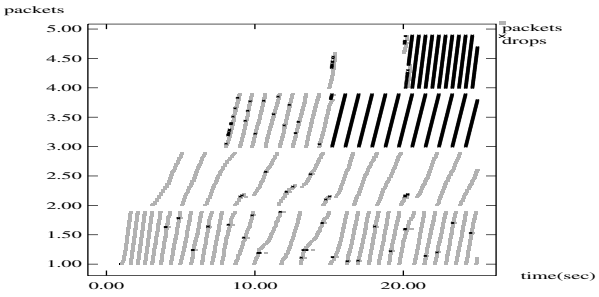


(a) sequence number of each flow

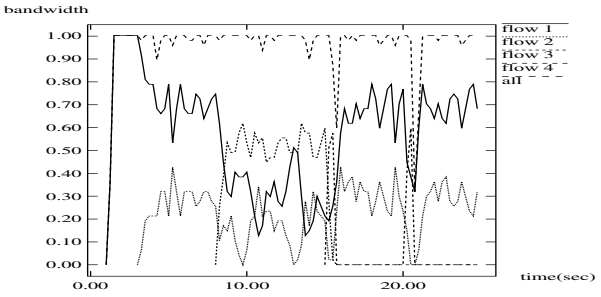


(b) bandwidth use of each flow

Figure 6: Test 1 with normal RED: unresponsive flows without flow-valve

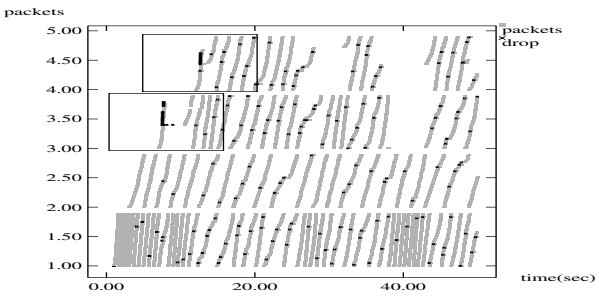


(a) sequence number of each flow

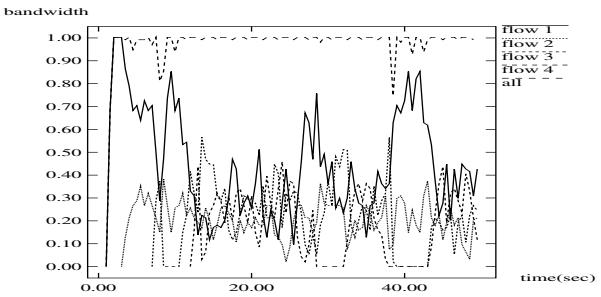


(b) bandwidth use of each flow

Figure 7: Test 1: unresponsive flows under flow-valve

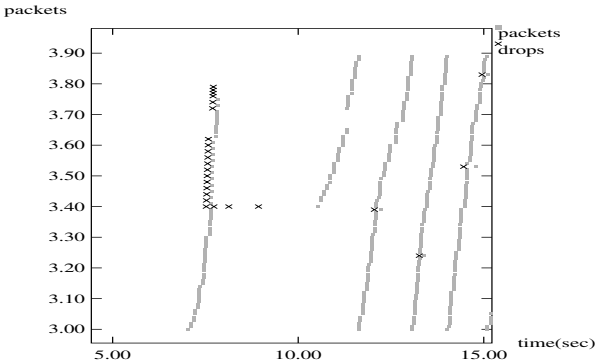


(a) sequence number of each flow

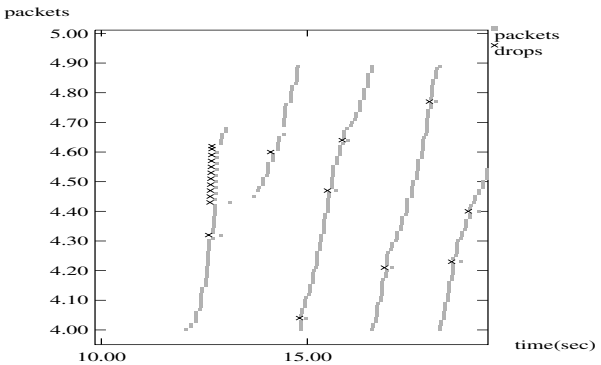


(b) bandwidth use of each flow

Figure 8: Test 2: 4 TCP sessions under flow-valve



(a) Flow 3: throttled by flow-valve



(b) Flow 4: normal timeout

Figure 9: Startup behaviors: flow-3 and flow-4 in Test 2

4.3 Test 2

Test 2 is a 50-second-long sequence that illustrates interaction among 4 TCP flows. Flow 1 and Flow 2 are same as in Test 1. Flow 3 and Flow 4 emulate on/off sources with a large window size.

Flow 1 ftp from S1 to S3 (20 segment window size)

Flow 2 ftp from S2 to S3 (5 segment window size)

Flow 3 ftp from S1 to S4 (40 segment window size)
off period: 38–43

Flow 4 ftp from S2 to S4 (40 segment window size)
off period: 28–32, 37–43

When Flow 3 starts at time 7, it loses a burst of packets during the first slow-start and the estimated packet drop rate of Flow 3 exceeds the threshold p_{th}. Flow 3 is detected as overpumping but freed after it backs off. Flow 3 is never detected as overpumping again since it sets $ssthresh$ at the first packet loss.

Figure 9 (a) shows the startup behavior of Flow 3. When the instantaneous queue length hits the limit, Flow 3 loses many packets and judged as overpumping. Flow 3 backs off exponentially and, at the 4th retransmission, the retransmission interval reaches the backoff threshold d_{th} and Flow 3 is freed.

At time 12, another flow, Flow 4, starts up and loses packets during the first slow-start as Flow 3 did. This time, the packet drop rate does not reach p_{th} and the flow-valve is not activated. Figure 9 (b) shows the startup behavior of Flow 4.

The difference between (a) and (b) in Figure 9 illustrates the impact of the backoff test to TCP. The penalty of the backoff test in (a) can be compared with a normal timeout in (b). The penalty of the normal timeout in (b) is smaller than the penalty of the flow-valve in (a) but they are in the same order. The difference will be smaller for most TCP implementations because of a larger minimum timeout value. It also suggests that, if a flow experiences multiple timeouts under heavy congestion, it is more harmful than a single penalty of the backoff test. It is also beneficial for a router; forcing exponential backoff is more effective in dissolving congestion than a few independent timeouts.

The packet loss at the first slow-start in this simulation is somewhat artificial by setting a small queue size limit. It is less likely to occur if the queue limit is big enough to absorb a transient increase of the instantaneous queue length. This setting is used to show a possible scenario in which a legitimate TCP is judged as overpumping and also to show the impact of the backoff test to a normal TCP.

The traffic pattern in Test 2 is highly dynamic but both the average queue length and the flow's packet drop rates are maintained within the proper range except the two first slow-starts mentioned above. It confirms that the flow-valve has no effect as long as RED works in the proper range.

5 Conclusion

As new transport protocols are emerging for audio/video or multicasting, end-to-end congestion control is increasing its importance for the evolution of the Internet. The current Internet, however, lacks a mechanism to raise awareness of the need for end-to-end congestion control.

We have shown the effects of TCP timeouts under heavy congestion, and proposed the flow-valve, a safety-valve for RED, that protects router resources in times of congestion at the cost of maintaining a small number of per-flow states. The flow-valve detects overpumping flows by a local decision and forces them to back off. Our design considerably simplifies an implementation of the penalty-box model.

The flow-valve provides a simple rule; if a flow loses too many packet but still using too much bandwidth, the flow is forced to back off. The flow-valve provides an incentive for end-to-end congestion control to keep the packet drop rate low under moderate congestion, and to conservatively back off under heavy congestion.

Our simulation results have demonstrated that the flow-valve keeps traffic within the control range of RED even in the face of misbehaving flows. Further, it helps isolate undesirable behavior of conformant TCP.

The flow-valve also has been successfully implemented onto FreeBSD as an extension to the ALTQ RED module [Cho98]. The flow-valve implementation on FreeBSD is about 600 lines in C and is included in the ALTQ release. We hope that the flow-valve will obtain field experience and provide an incentive in support of end-to-end congestion control.

References

[AP99] M. Allman and V. Paxson. On Estimating End-to-End Network Path Properties. In Proceedings of *SIGCOMM99*, 263–274, Cambridge, MA, September 1999.

[Cho98] K. Cho. A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers. In Proceedings of *USENIX 1998 Annual Technical Conference*, New Orleans, LA, June 1998.

[Cho99] K. Cho. Flow-valve: Embedding a Safety-valve in RED. Technical Report, 1999. ftp://ftp.csl.sony.co.jp/pub/kjc/papers/fv99.ps.gz

[FJ92] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[FJ93] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transaction on Networking*, 1(4):397–413, August 1993.

[FF99] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transaction on Networking*, August 1999.

[FFT98] S. Floyd, K. Fall and K. Tieu. Estimating Arrival Rates from the RED Packet Drop History. *Draft paper*, April 1998.

[Floyd91] S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic. *Computer Communication Review*, 21(5):30–47, October 1991.

[Jacobson88] V. Jacobson. Congestion Avoidance and Control. *Computer Communication Review*, 18(4)314–329, August 1988.

[Jacobson90] V. Jacobson. Berkeley TCP Evolution from 4.3-Tahoe to 4.3-Reno. In Proceeding of *the Eighteenth Internet Engineering Task Force*, p.365, September 1990.

[Kumar98] A. Kumar. Comparative Performance Analysis of Versions of TCP in Local Network with a Lossy Link. *IEEE/ACM Transactions on Networking*, 6(4), August 1998.

[LM97] D. Lin and R. Morris. Dynamics of Random Early Detection. In Proceedings of *SIGCOMM97*, 127–137, Cannes, France, September 1997.

[LU97] T. V. Lakshman and U. Madhow. The Performance of Networks with High Bandwidth-delay Products and Random Loss. *IEEE/ACM Transactions on Networking*, June 1997.

[MF95] S. McCanne and S. Floyd. NS (Network Simulator). http://www-nrg.ee.lbl.gov/ns/, 1995.

[MSMO97] M. Mathis, J. Semke, J. Mahdavi and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *Computer Communication Review*, 27(3):67–82, July 1997.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In Proceedings of *SIGCOMM98*, 303–314, Vancouver, Canada, September 1998.