Packet Dropping Mechanisms: Some Examples and Analysis

Rong Pan, Chandra Nair, Brian Yang, Balaji Prabhakar
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
{rong,mchandra,balaji}@stanford.edu

Abstract

Recently, there have been attempts to design packet dropping mechanisms for sharing the bandwidth at a congested link fairly among the flows using this link. Such mechanisms would allow the use of a simple FIFO buffer at the egress link of a router, instead of the more complicated buffer structures required by algorithms like Fair Queueing. These various packet dropping mechanisms provide different degrees of fairness and incur different levels of implementation complexity. Their performance is generally evaluated by simulation. This paper attempts to develop theoretical models for analyzing the trade-off between fairness and complexity. We begin by considering some very simple packet dropping mechanisms, which capture the essential features of the problem, and go on to build more accurate theoretical models using fluid analysis. A previously-introduced algorithm, called CHOKe, is analyzed in detail. The accuracy of the fluid model is also verified using simulations.

I. Introduction

The Internet provides a connectionless, best effort service using the Internet Protocol (IP). It relies on end hosts to regulate their instantaneous offered traffic during periods of congestion. Given the increasing variety of transport layer protocols, and the diverse requirements of applications, there is a need for network routers to allocate bandwidth fairly. But fairness can be expensive to provide: the well-known Fair Queueing (FQ) algorithm [1] provides packet-by-packet fairness but requires per-flow state and per-flow queueing, which can be expensive to implement.

Recently, a collection of algorithms have been proposed for providing fairness by differentially dropping packets, rather than by differentially queueing and scheduling them as in FQ. Such packet dropping schemes are generally based on "active management schemes", such as the Random Early Detection (RED) algorithm [2]. These differential dropping algorithms organize the output or egress buffer of a router as a simple FIFO queue, and by dropping packets belonging to the different flows in different proportions, they aim to allocate the outgoing link's bandwidth fairly among these flows. Although these differential dropping mechanisms usually require a small data structure, in addition to the FIFO buffer, to guide dropping decisions, the extent of fairness that they can achieve can be substantial.

For our purposes, we shall classify packet dropping schemes into three categories, depending on the type of information they use to drop packets: (i) size-based schemes, (ii) history-based schemes, and (iii) content-based schemes. We briefly describe these schemes now, and will elaborate later in Section II. Size-based schemes drop packets based on the current *size* of the FIFO packet buffer. They do not use any information about the *constitution* of the FIFO buffer, i.e. the numbers of packets belonging to different flows that currently inhabit the buffer. Nor do they use any information of the number of past arrivals and packets dropped from individual flows. History-based schemes may take into account both the current size of the FIFO buffer and the history of past arrivals and/or packet drops from individual flows. Finally, content-based schemes only take into account the current buffer-size and its *instantaneous constitution*; they do not use any history of past arrivals or packet drops. Note that size-based schemes drop packets non-preferentially while history-based and content-based schemes preferentially drop packets of individual flows to achieve fairness.

Our aim in this paper is to develop some very simple queueing models to understand the effectiveness of these schemes in providing a fair bandwidth allocation, and the attendant complexity of implementing them. The models we study in Section III are caricatures: they are meant to be very simple, and represent a first attempt at understanding the trade-off between fairness and implementation complexity. We shall see that, despite their simplicity, they reveal some interesting features of actual packet dropping schemes.

In Section IV we analyze the CHOKe algorithm in detail using a fluid model. The goodput obtained by a source as predicted by the theoretical fluid model is compared with simulations performed in ns [8]. These simulations show that the fluid model is very accurate. The model also provides insight into the degree of fairness achieved as the number of samples used by CHOKe is increased.

II. A CLASSIFICATION OF PACKET DROPPING SCHEMES

As mentioned in the introduction, we classify packet dropping mechanisms into three categories and study a very simple common model for evaluating them. We proceed by making some definitions.

The packet dropping schemes we are interested in all organize the egress buffer as a single FIFO queue containing all the packets from all the flows. Packets may be dropped from any location of the buffer, and the decision to drop a packet may be based on one of the following criteria:

1. The current size of the FIFO buffer. A packet may be dropped either deterministically or according to a probability which depends on the size of the buffer. The main examples of such a packet dropping mechanism are DropTail and RED. We shall refer to schemes of this type as *size-based schemes*.
2. A history of past arrivals and/or dropped packets. In this case we assume that in addition to the current buffer size, a packet dropping mechanism maintains a tally of the number of packets received and/or the number of packets dropped from each flow. The duration over which such tallies (or histories) may be maintained is a parameter of the dropping mechanism. A packet of a certain flow is dropped by referring to that flow's history, but not by referring to the contents of the current queue-size. Examples in this category are RED with penalty box [3], SRED [5], AFD [7] and RED-PD [4]. These schemes will be referred to as *history-based schemes*.
3. The current contents of the FIFO buffer. In this case the packet dropping mechanism is allowed to know only the number of packets belonging each flow that are *currently* in the queue, and it drops packets using only this information. It does not maintain any history. The CHOKe algorithm [6] is an example of this type of scheme, which we refer to as a *content-based scheme.*

III. A SIMPLE MODEL OF PACKET DROPPING SCHEMES

Consider a single flow, and suppose that its packets arrive at a queue. We assume that the inter-arrival time is IID, with an arbitrary distribution on the marginals. Let the mean inter-arrival time be λ^{-1}. The queue has a limited buffer space, say equal to B ($0 \leq B < \infty$).

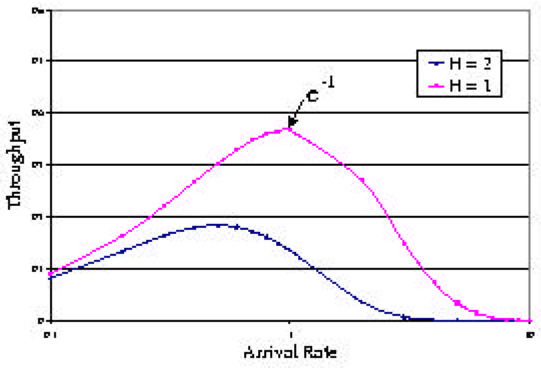

Fig. 1. A flow's throughput in the history-based schemes

Arriving packets are either dropped on arrival, or enqueued at this buffer and possibly dropped later depending on the dropping discipline. Suppose each packet requires a fixed S amount of service time ($0 \leq S < \infty$). Using this very simple model, we wish to understand the performance of each of the three different types of packet dropping schemes.

Size-based schemes: Consider the simple case of $B = 0$ and $S = 0$. Packets arrive at rate λ and are either served instantly or dropped, based on the outcomes of independent coin flips. If p is the probability that a packet is served, then the goodput λ_g is equal to λp packets/sec. As λ grows, this grows without bound.

If $B > 0$ and $S = 1$, say, and packets arriving to a full buffer are dropped, then the goodput also increases with the arrival rate λ and will saturate at 1 as $\lambda \to \infty$. A similar conclusion will hold if arriving packets are dropped with an increasing probability as the congestion-level of the buffer increases.

Thus, as might be expected, size-based schemes are unable to limit the goodput obtained by a flow to a value less than the service rate. In particular, a source can consume all of the server's capacity by increasing its sending rate.

History-based schemes: Consider the queue with $B = 0$ and $S = 1$. There is only room for one packet – the one in service – at any given time. This packet is served for 1 unit of time. Separately, let there also be a "history clock" which is either in the OFF state or in the ON state. Arriving packets are admitted into the queue only if the history clock is in the OFF state. Otherwise, they are dropped. As soon as a packet arrives, the clock is set to the ON state and stays there for H time units. Note that this operation is triggered by every arriving packet, whether it is admitted or dropped.

Now, it is easy to see that the only packets that are served (i.e. not dropped) are those which arrive at least H time units after the previous packet. This fact makes it easy to compute the goodput given the inter-arrival time distribution. For Poisson arrivals of rate λ, the goodput is $\lambda e^{-\lambda H}$. As the sending rate of the source increases, the goodput increases to a maximum of $(He)^{-1}$ and decreases to 0 as λ increases further. Also note that the maximum value of the goodput decreases as H increases. Figure 1 plots the goodput as a function of λ for $H = 1$ and $H = 2$.

Thus history-based schemes, which use some information about past arrivals to decide whether to drop arriving packets, are able to limit the amount of goodput obtainable by a single source to a value strictly less than the service rate.

Content-based schemes: Now, consider again the queue with $B = 0$ and $S = 1$. An incoming packet is processed immediately if the server is idle (i.e. there is no other packet being served).

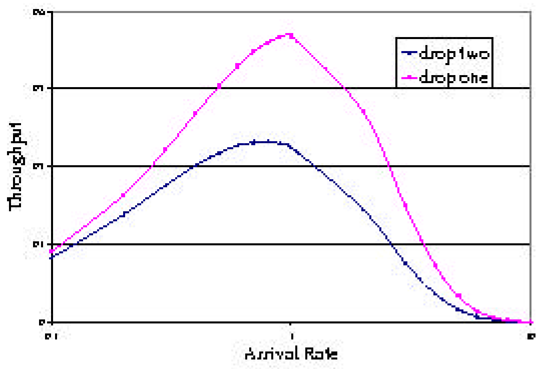

Fig. 2. A flow's goodput in the content-based scheme

Otherwise, the system can discard either (i) the packet in service, or (ii) both the incoming packet and the one in service.

In the former case, it is obvious that the only packets that can successfully depart are those which arrive at least one unit of time before their followers. Assuming Poisson arrivals of rate λ, it is easy to compute that a flow's goodput equals to $\lambda e^{-\lambda}$.

In the latter case where both packets are dropped, in order for a packet to depart, it must satisfy two criteria. First, it must arrive at an empty system to avoid a collision with another packet. Second, no other packet can arrive while it is being served. For Poisson arrivals of rate λ, it can be shown that the goodput of a flow equals

$$\frac{\lambda e^{-\lambda}}{2 - e^{-\lambda}}. \tag{1}$$

Figure 2 shows a flow's goodput in each of the two cases considered. Note that based solely on the packet constitution in the system, and without using any history, the content-based schemes described can limit the goodput of a flow to less than the service capacity. When $B > 0$, it is more cumbersome to analyze this scheme using the simple model described here. We will describe a more powerful technique in the following section using CHOKe as a concrete example.

IV. A Fluid Model Analysis of CHOKe

In this section, we study the behavior of a content-based scheme, CHOKe [6], using a fluid model. The CHOKe algorithm, like the other packet dropping schemes, uses a FIFO buffer to store packets before transmitting them on the outgoing link. The precise description of the algorithm is provided in earlier work (see [6]). For the purposes of developing the fluid model, we describe it here more simply as follows.

When a packet arrives, another packet is drawn uniformly at random from the FIFO buffer, and its header is compared with that of the arriving packet. If both packets belong to the same flow, then they are both dropped. Otherwise, the sampled packet is replaced in its original position in the FIFO buffer and the arriving packet is queued at the tail of the FIFO buffer. Multiple packets can also be drawn independently and uniformly at random from the buffer for comparison. We will analyze this version of CHOKe later.

The FIFO buffer is modeled as a permeable tube through which packets of the different flows, modeled as distinct types of fluid, pass. When a fluid unit from flow i arrives at the input of the tube, a unit amount of fluid is chosen from the tube uniformly at random. If the fluid chosen from the tube is of the same type as the arriving fluid, both the incoming fluid and the fluid from the tube are released from the system. In the real algorithm, the incoming packet is entirely dropped or retained. However, here it is possible to drop fractions of packets, since fluid units can be defined arbitrarily. We can see this as a time-averaged version of what the CHOKe algorithm does. Thus, on average, the rate at which flow i packets are dropped from the buffer is equal to the product of the average rate of flow i arrivals, equal to λ_i, say, and the probability that a packet from flow i is drawn from the queue.

A. CHOKe with one sample

In this subsection, we look at the version of the CHOKe algorithm in which only one sample is drawn from the buffer for comparison. Let F be the total amount of fluid in the tube, and F_i be the amount of flow i fluid in the buffer. The fraction of type i fluid in the tube, F_i/F, is denoted by p_i. Define $L_i(t)dt$ to be the amount of flow i fluid that is t time units old, i.e. $L_i(t)dt$ represents the amount of flow i fluid that arrived t time units ago and remains in the tube at the present time. Thus, $L_i(0)$ is the rate at which type i fluid entered the tube at this moment, which equals the raw arrival rate λ_i minus the fraction of fluid that was dropped at the ingress of the tube. Since p_i is the fraction of type i fluid present in the FIFO buffer, the chance that a randomly chosen fluid unit is of type i equals p_i. This will cause arriving type i fluid drops at rate $\lambda_i p_i$. Therefore, $L_i(0) = \lambda_i(1 - p_i)$.

After dt period, the amount of fluid that arrived $t + dt$ time ago and still remains in the tube is $L_i(t + dt)dt$. The difference, $L_i(t)dt - L_i(t + dt)dt$, is the amount of fluid dropped from the tube during the infinitesimal period dt. The drop is due to the occurrence of two events: (i) a unit of fluid belonging to the $L_i(t)dt$ volume is sampled from the tube, and (ii) incoming fluid of type i was compared against this sampled fluid. Since in the period dt, on average, $\lambda_i dt$ type i fluid arrives, we get

$$\begin{aligned} L_i(t)dt - L_i(t + dt)dt &= \lambda_i dt \frac{L_i(t)dt}{F} \\ -\frac{dL_i(t)}{dt} &= \frac{\lambda_i L_i(t)}{F} \\ \text{and so,}\ \ L_i(t) &= L_i(0)e^{-\lambda_i t/F} = \lambda_i(1 - p_i)e^{-\lambda_i t/F}. \end{aligned} \tag{2}$$

Equation (2) shows that once fluid is admitted into the pipe, it decays exponentially over time. It also shows that flows with a higher arrival rates die out faster.

We are now left with determining p_i. We note that since drops always occur in pairs, the throughput of flow i is $\mu_i = \lambda_i(1 - 2p_i)$. A flow-conservation argument now implies

$$p_i = \frac{\lambda_i - \mu_i}{2\lambda_i}. \tag{3}$$

Assuming the expected queueing delay in the FIFO is D, we obtain

$$\mu_i = L_i(D) = (1 - p_i)\lambda_i e^{-\lambda_i D/F} = \frac{(\lambda_i - \mu_i)e^{-\lambda_i D/F}}{2}.$$

Solving for μ_i yields

$$\mu_i = \frac{\lambda_i e^{-\lambda_i D/F}}{2 - e^{-\lambda_i D/F}}. \tag{4}$$

We pause to note that equations (1) and (4) have a strikingly similar form! The notable difference is due to the delay of D units introduced by the FIFO buffer, which is common to all the flows sharing the FIFO buffer. In fact, this difference can be viewed as the result of the FIFO buffer maintaining a common *history* of all the packets that arrived within the past D units of time. Thus, the operations in CHOKe are the same as those in the simple content-based scheme discussed in Section III with the FIFO buffer providing a recent history of arrivals.

Note that D/F is independent of a particular flow, but depends on the total amount of traffic input to the FIFO buffer. Hence, for a fixed rate of aggregate input traffic, a flow's throughput stays the same no matter what the remaining traffic mix is. The constant D/F can be obtained by setting the net departure rate from the congested FIFO buffer to equal the service capacity (normalized to 1 here):

$$\sum_{i=1}^{N_f} \mu_i = 1, \tag{5}$$

where N_f is the number of flows sharing the FIFO buffer. Further, since the congested buffer cannot be empty, the following condition must be satisfied

$$\sum_{1}^{N_f} p_i = 1. \tag{6}$$

Using equations (3), (4), (5) and (6), one can solve explicitly for the throughput of each flow i under the CHOKe algorithm when one sample is taken from the FIFO buffer.

B. CHOKe with multiple samples

When multiple packets are sampled from the FIFO buffer, and all packets belonging to the same flow as the arriving packet are dropped, we may easily write down differential equations as above. The details follow.

Let M be the number of packets drawn from the queue. Using similar arguments to those in Subsection IV-A, we easily obtain

$$-\frac{dL_i(t)}{dt} = \frac{\lambda_i M L_i(t)}{F}. \tag{7}$$

Thus, $L_i(t) = L_i(0)e^{-M\lambda_i t/F}$.

A packet may be admitted and placed at the tail of the FIFO buffer iff it survives M comparisons, one each against the randomly drawn samples. As before let p_i be the probability of drawing a flow i packet from the queue. Then the incoming packet is admitted with probability $(1-p_i)^M$. Therefore, $L_i(0) = \lambda_i(1-p_i)^M$, and the throughput of flow i, μ_i, equals

$$\mu_i = L_i(D) = \lambda_i(1-p_i)^M e^{-\frac{M\lambda_i}{F}D}. \tag{8}$$

Flow conservation tells us that $L_i(0) - \mu_i = \lambda_i M p_i$, and so

$$(1-p_i)^M(1-e^{-\frac{M\lambda_i}{F}D}) = Mp_i. \tag{9}$$

Equations (5) and (6) continue to be hold in the multiple sample case, and together with equations (8) and (9) they allow one to solve for μ_i explicitly.

C. Applications of the CHOKe fluid model

In this subsection, we apply the results obtained above to study traffic mixes of UDP and TCP sources arising at a congested router in the Internet. UDP sources send data at a fixed rate regardless the level of congestion in the network while the data rate of TCP sources are responsive to congestion, and hence to packet drops. The fluid model developed in the previous section applies to UDP sources and we consider TCP sources in steady state, in which case their (congestion-sensitive) data rates are assumed to be so as to fill up the capacity left by unconsumed by the UDP sources.

1) One UDP source, multiple TCP sources: There are one UDP source and N_t identical TCP flows competing at a bottlenecked link of unit capacity, where N_t is assumed to be large. First suppose that CHOKe draws one one packet at random from the FIFO buffer. The UDP source being non-responsive offers traffic at rate λ_u, regardless of the level of congestion. Since TCP sources are congestion aware, they send traffic at a rate allowed by the router, and the rate is indicated to them by the dropping of packets. As a result, they incur much less packet drops compared to the unresponsive UDP flow. This allows us to assume that the throughput of a TCP flow is roughly equal to its arrival rate [1]. In other words, if each TCP source offers traffic at λ_t in steady state, then $\mu_t \approx \lambda_t$. In such a network setup, we wish to study the performance of the system under different UDP loads.

By Little's formula, the expected number of packets in the buffer that belong to a single TCP flow is the product of the expected delay D and its offered rate λ_t. Letting p_t to equal the fraction of packets belonging to a TCP flow, and again assuming that there are a total of F packets in the FIFO buffer, Little's formula yields

$$Fp_t = D\lambda_t = D\mu_t, \tag{10}$$

where the second equation follows from our assumption that the arrival and departure rates for TCP flows are roughly equal.

Let p_u be the fraction of packets belonging to the UDP flow. Using equation (3) this may be written as

$$p_u = \frac{\lambda_u - \mu_u}{2\lambda_u}.$$

Substituting for p_t and p_u in (6) gives

$$\frac{DN_t\mu_t}{F} + \frac{\lambda_u - \mu_u}{2\lambda_u} = 1 \Rightarrow \frac{DN_t\mu_t}{F} = \frac{\lambda_u + \mu_u}{2\lambda_u}. \tag{11}$$

Now from (5) we get that $N_t\mu_t = 1 - \mu_u$. Therefore, equation (11) becomes

$$\frac{D}{F} = \frac{\lambda_u + \mu_u}{2(1-\mu_u)\lambda_u}. \tag{12}$$

Substituting for D/F in (4), we obtain a relationship between λ_u and μ_u:

$$exp\left(\frac{\lambda_u + \mu_u}{2(1-\mu_u)}\right) = \frac{\lambda_u + \mu_u}{2\mu_u}. \tag{13}$$

[1] This statement is not valid in general, but holds in the scenarios of interest to us: i.e. when there are UDP flows present and account from the largest fraction of dropped packets.

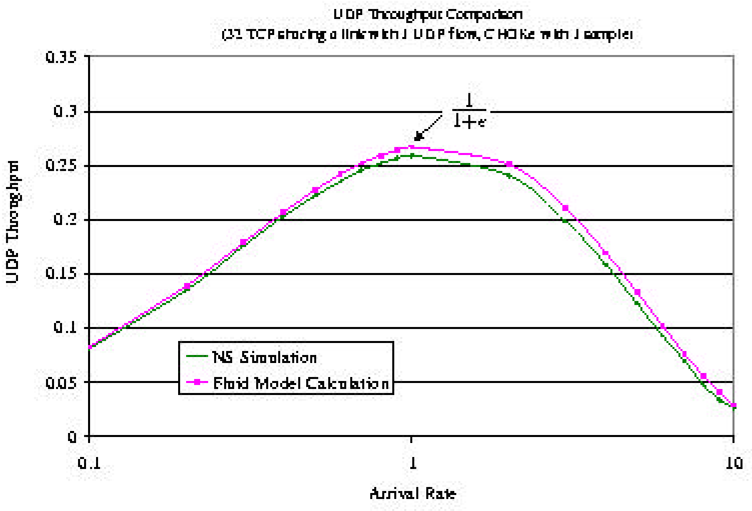

Fig. 3. μ_u vs λ_u (1 UDP with 1 sample)

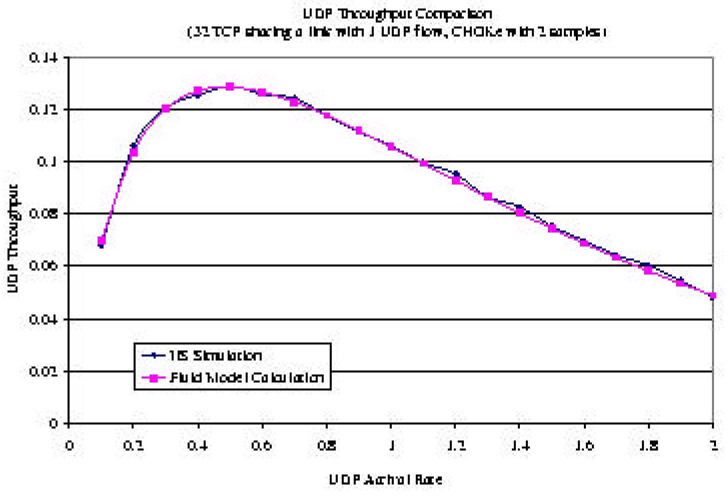

Fig. 4. μ_u vs λ_u (1 UDP with 2 samples)

Figure 3 uses equation (13) to plot μ_u as a function of λ_u, and compares this against simulations obtained using ns. The ns simulations use 32 TCP sources, 1 UDP source sharing a congested link of capacity equal to 1 Mbps (normalized here to 1). It demonstrates that the fluid model is in very close agreement with the ns simulation results. Using (13) one can determine the maximum throughput obtained by the UDP source to be $\mu_u^{max} = (1+e)^{-1} \approx 0.269$ Mbps. Further, this is achieved when its sending rate equals $\lambda_u^{max} = (2e-1)(1+e)^{-1} \approx 1.193$ Mbps. Thus, when it is obtaining its highest possible share of the congesting link's bandwidth, the UDP sources looses $(1.193 - 0.269)/1.193 = 77.45\%$ of its packets.

What happens when two packets are drawn at random from the queue? From flow conser-

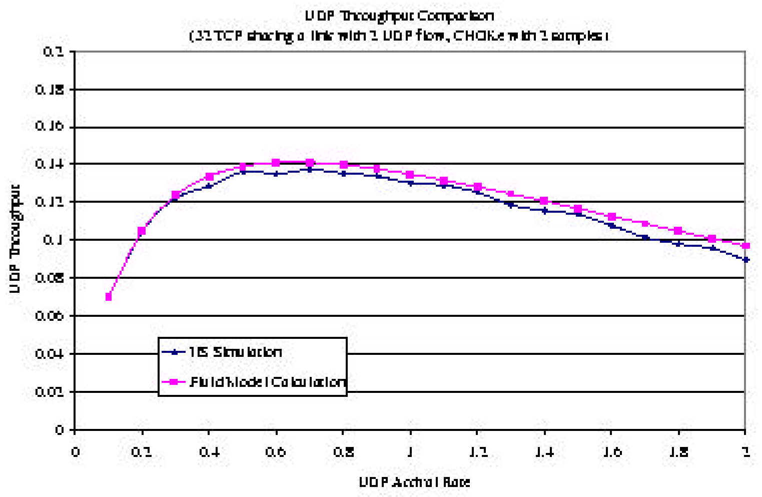

Fig. 5. μ_u vs λ_u (2 UDP with 2 samples)

vation, we know that

$$\lambda_u(1-p_u)^2 - \mu_u = 2\lambda_u p_u \;\;\Rightarrow\;\; p_u = 2 - \sqrt{3 + \frac{\mu_u}{\lambda_u}}. \tag{14}$$

We also assume $\mu_t \approx \lambda_t$ and use the fact that $p_t = \lambda_t D/F$. Using these at Equations (5) and (6) yields $D/F = (1-p_u)(1-\mu_u)^{-1}$. And the relationship between μ_u and λ_u is:

$$\mu_u = \lambda_u \left(\sqrt{3+\frac{\mu_u}{\lambda_u}} - 1\right)^2 exp\left(-\frac{2\lambda_u}{1-\mu_u}\left(\sqrt{3+\frac{\mu_u}{\lambda_u}}-1\right)\right) \tag{15}$$

Figure 4 plots the above relationship in comparison with an ns simulation. The plot demonstrates the accuracy of the theory and also points out the fact that by using an extra sample from the queue, CHOKe is able to further limit the maximum throughput obtained by the UDP source.

2) Multiple UDP sources, multiple TCP sources: Now suppose that N_t TCP flows and N_u UDP flows share a link of unit capacity. For ease of exposition we assume that all the UDP flows send at equal rates and all TCP flows continue to respond identically to congestion.

Consider the concrete case of $N_u = 2$. Let both UDP flows have an arrival rate λ_u and let all TCP flows have an arrival rate of λ_t. As before, assume that TCP sources incur small drops, hence $\mu_t \approx \lambda_t$, and use the fact that $p_t = \lambda_t D/F$. From (5) and (6) we obtain $D/F = (1-2p_u)(1-2\mu_u)^{-1}$ and find the following relation between μ_u and λ_u

$$\mu_u = \lambda_u \left(\sqrt{3+\frac{\mu_u}{\lambda_u}} - 1\right)^2 exp\left(-\frac{2\lambda_u}{1-\mu_u}\left(2\sqrt{3+\frac{\mu_u}{\lambda_u}}-3\right)\right). \tag{16}$$

Figure 5 shows a plot of the above relationship as λ_u increases and, once again, we see that the fluid model matches well with ns simulation.

Now, consider a more general system in which N_u and N_t are variables and suppose M random samples are taken from from the queue. It is interesting to determine, for a given value of N_u, how large M should be so as to limit the throughputs obtained by the UDP sources to less than the service rate. It is clear that M should increase with N_u, but what relationship governs this increase?

Set $\beta = MD/F$ and assume $\mu_t \approx \lambda_t$. Then from (9) we have

$$(1 - p_u)^M - Mp_u \geq 0. \tag{17}$$

Notice that the left hand side is a monotonically decreasing function of p_u in the range (0,1). It may be shown that we require $M > N_u(2 - \sqrt{2})$ in order for (17) to be valid. Now, if $\lambda_u \to \infty$, then $\beta\lambda_u \to \infty$. Since $xe^{-x} \to 0$ as $x \to \infty$, we deduce from (8) that $\mu_u \to 0$. Thus, $(2 - \sqrt{2})N_u$ random samples suffice to ensure that $\mu_u \to 0$ as $\lambda_u \to \infty$. This analysis tells us that in order to limit the throughput obtained by multiple unresponsive or aggressive flows in a network, one needs only to sample a number of packets of the order of unresponsive flows. When the majority of the Internet traffic is TCP-friendly (i.e. congestion-responsive), this simple analysis suggests that a content-based scheme can be much simpler to implement than history-based schemes, which usually require state of the order of the number of flows.

V. Conclusions

We have analyzed three types of packet dropping schemes: size-based, history-based and content-based. A simple model revealed the role played by preferential dropping mechanisms that use information from a history of past arrivals/drops or from the current contents of the FIFO buffer in limiting the throughput obtainable by sources that do not respond to congestion. We also developed a quite accurate fluid model of a previously-proposed packet dropping scheme: CHOKe. This model generalizes in several directions and informs that the number of samples required to limit the throughput obtained by congestion unresponsive flows is on the order of their number.

References

[1] Demers, A., Keshav, S. and Shenker, S., "Analysis and simulation of a fair queueing algorithm", *Journal of Internetworking Research and Experience*, Oct. 1990.

[2] Floyd, S. and Jacobson, V., "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transaction on Networking*, 1(4), pp 397-413, Aug. 1993.

[3] Floyd, S., and Fall, K., "Router Mechanisms to Support End-to-End Congestion Control", *LBL Technical report*, February 1997.

[4] Mahajan, R. and Floyd, S., "Controlling High-Bandwidth Flows at the Congested Router" *to appear ICNP 2001*

[5] Ott, T., Lakshman, T. and Wong, L., "SRED: Stabilized RED", *Proceedings of INFOCOM'99*, March 1999.

[6] Pan, R., Prabhakar, B. and Psounis, K., "CHOKe: A stateless active queue management scheme for approximating fair bandwidth allocation", *Proceedings of INFOCOM'2000*.

[7] Pan, R. Breslau, L., Prabhakar, B. and Shenker, S. "Approximate Fairness Throughput Differential Dropping", *submitted, a summary was presented at Poster Session of ACM SIGCOMM'2001*.

[8] ns - Network Simulator (Version 2.0), October 1998.