Understanding CHOKe

Ao Tang Jiantao Wang Steven H. Low
California Institute of Technology, Pasadena, USA
{aotang@, jiantao@cds., slow@}caltech.edu

***Abstract*— A recently proposed active queue management, CHOKe, is stateless, simple to implement, yet surprisingly effective in protecting TCP from UDP flows. As UDP rate increases, even though the number of UDP packets in the queue rises, its bandwidth share eventually drops to zero, in stark contrast to the behavior of a regular FIFO buffer. We derive a detailed model of CHOKe that accurately predicts this, and other behaviors of CHOKe, and validate the model with simulations. Its key features are the incorporation of the feedback equilibrium of TCP with dropping probability and the spatial characteristics of the queueing process. CHOKe produces a "leaky buffer" where packets can be dropped as they move towards the head of the queue. This leads to a spatially non-uniform distribution of packets and their velocity, and makes it possible for a flow to simultaneously maintain a large number of packets in the queue and receive a vanishingly small bandwidth share. This is the main mechanism through which CHOKe protects TCP from UDP flows.**

I. Introduction

TCP is believed to be largely responsible for preventing congestion collapse while Internet has undergone dramatic growth in the last decade. Indeed, numerous measurements have consistently shown that more than 90% of traffic on the current Internet are still TCP packets, which, fortunately, are congestion controlled. Without a proper incentive structure, however, this state of affair is fragile and can be disrupted by the growing number of non-rate-adaptive (e.g., UDP-based) applications that can monopolize network bandwidth to the detriment of rate-adaptive applications. This has motivated several active queue management schemes, e.g., [13], [3], [7], [18], [15], [17], [2], that aim at penalizing aggressive flows and ensuring fairness. The scheme, CHOKe, of [17] is particularly interesting in that it does not require any state information and yet can provide a minimum bandwidth share to TCP flows. In this paper, we provide a detailed analytical model of CHOKe, and discuss some of its implications.

The basic idea of CHOKe is explained in the following quote from [17]:

> When a packet arrives at a congested router, CHOKe draws a packet at random from the FIFO (first-in-first-out) buffer and compares it with the arriving packet. If they both belong to the same flow, then they are both dropped; else the randomly chosen packet is left intact and the arriving packet is admitted into the buffer with a probability that depends on the level of congestion (this probability is computed exactly as in RED).

The surprising feature of this extremely simple scheme is that it can bound the bandwidth share of UDP flows regardless of their arrival rate. In fact, as the arrival rate of UDP packets increases without bound, their bandwidth share approaches zero! An intuitive explanation is provided in [17]: "the FIFO buffer is more likely to have packets belonging to a misbehaving flow and hence these packets are more likely to be chosen for comparison. Further, packets belonging to a misbehaving flow arrive more numerously and are more likely to trigger comparisons." As a result, aggressive flows are penalized. This however does not explain why a flow that maintains a much larger number of packets in the queue does not receive a larger share of bandwidth, as in the case of a regular FIFO buffer. It turns out that a precise understanding of this phenomenon requires a detailed analysis of the queue dynamics, the key feature of our model. A simple model of CHOKe is also presented in [17] that assumes a Poisson packet arrival process and exponential service time. The Poisson assumption is critical in order to use the PASTA (Poisson Arrival Sees Time Averages) property to compute drop probabilities. This model however ignores both the feedback equilibrium of the TCP/CHOKe system and the spatial characteristics of the queue.

Here, we adopt a deterministic fluid model that explicitly models both features (Section II).Our model predicts, and simulation confirms, that as UDP rate becomes large, not only does the total number of UDP packets in the queue increase, more importantly, the *spatial* distribution of packets becomes more and more concentrated at the tail of the queue, and drops rapidly to zero towards the head of the queue. Hence even though the total number of UDP packets in the queue is large, all of them will be dropped before they advance to the head. As a result the UDP bandwidth share drops to zero, in stark contrast to a non-leaky FIFO queue where UDP bandwidth shares approaches 1 as its input rate increases without bound.

Our current model is too complex to be solved analytically. We outline a numerical solution (Section II-F), and compare our numerical results with detailed ns-2 simulations. They match accurately not only with average behavior of CHOKe as reported in [17], but also with much finer spatial characteristics of the queueing process.

The technique presented here should be applicable to analyzing the queueing process in other types of leaky buffer.

II. CHOKe model

The basic idea of CHOKe is summarized in Section I. A flow chart describing its implementation with RED is given in Figure 1 from [17].

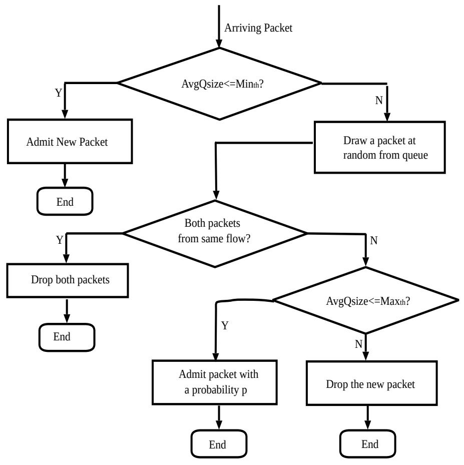


Fig. 1. CHOKe flow chart

In general, one can choose more than one packet from the queue, compare all of them with the incoming packet, and drop those from the same flow. This will improve CHOKe's performance, especially when there are multiple unresponsive sources. It is suggested in [17] that more drop candidate packets be used as the number of unresponsive flows increases. Here, we focus on the modelling of a single drop candidate packet. The analysis can be extended to the case of multiple drop candidates.

The general setup for our model and our simulations is shown in Figure 2.

There is a single bottleneck link with capacity c packets per second. We focus on the FIFO buffer at router R1 where packets are queued. The buffer is shared by N identical TCP flows and a single UDP flow.All TCP flows have a common round trip delay of d seconds.We assume the system is stable and model its equilibrium behavior.

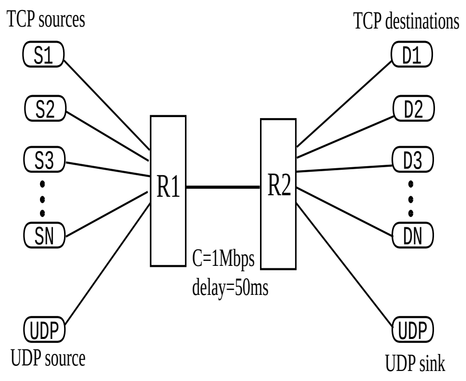


Fig. 2. Network topology

There are two main components of our model; the first models the feedback loop between TCP flows and the dropping probability, and the other deals with the detailed dynamics of a "leaky queue". Our main goal is to understand how bandwidth shares of UDP and TCP flows vary with (the uncontrolled) UDP rate and the number of TCP flows.

A. Notations

Quantities (rate, backlog, dropping probability, etc) associated with the UDP flow are indexed by 0. Those associated with TCP flows are indexed by $i = 1, \ldots, N$; since the TCP sources are identical, these quantities all have the same value, and hence we will usually refer to flow 1 as the generic TCP flow.

We collect here the definitions of all the variables and some of their obvious properties:

b_i: packet backlog from flow i, $i = 0, \ldots, N$; $b_i = b_1, i \geq 1$.

b: total backlog; $b = b_0 + b_1 N$.

r: RED dropping probability:

$$r = \begin{cases} k(b - \underline{b}) & \text{if } \underline{b} \leq b \leq \overline{b} \\ 0 & \text{if } b \leq \underline{b} \\ 1 & \text{if } b \geq \overline{b} \end{cases} \quad (1)$$

where $k := p\_max/(\overline{b} - \underline{b})$ is the slope of the RED dropping profile.

h_i: The probability of incoming packets being dropped by choke for flow i, $i = 0, \ldots, N$:

$$h_i = \frac{b_i}{b} \quad (2)$$

p_i: overall probability that packets of flow i is dropped before it gets through, either by CHOKe or RED:

$$p_i = 2h_i + (1 - h_i)\, r \quad (3)$$

The explanation of (3) is provided below.

x_i:	source rate of flow i, $i = 0, \dots, N$.
τ:	queueing delay (not including the common propagation delay is d).
μ_i:	bandwidth share of flow i, $i = 0, \dots, N$; $\mu_0 + \mu_1 N = 1$, $\mu_i = \mu_1, i \geq 1$.
$\rho_i(y)$:	probability that the packet at position $y \in [0, b]$ in the queue belongs to flow i, $i = 0, \dots, N$.
$v(y)$:	velocity at which the packet at position $y \in [0, b]$ in the queue moves towards the head of the queue.

As mentioned above, by symmetry, $b_i = b_1$, $h_i = h_1$, etc, for $i \geq 1$. RED does not distinguish between different flows and hence its dropping probability r is common for all flows i. CHOKe on the other hand differentiates between TCP and UDP flows and hence h_0 and h_1 are generally different.

We make two remarks. First, It is important to keep in mind that x_0 is the only independent variable; all other variables listed above are functions of x_0, though this is not made explicit in the notation. Second, x_i is the sending rate of flow i. The rate at which flow i packets enter the tail of the queue (after going through CHOKe and RED on arrival) is $x_i(1-h_i)(1-r)$, and the rate at which flow$-i$ packets exit the queue (throughput) is $x_i(1-p_i) = \mu_i c$. Clearly, $x_i(1-p_i) \leq x_i(1-h_i)(1-r) \leq x_i$. Our primary goal is to understand how throughput $\mu_i c$ varies with x_0. As we show now, this turns out to require detailed understanding of the queueing process in a "leaky queue".

B. *TCP/CHOKe feedback loop*

Recent studies have shown that *any* TCP congestion control algorithm can be interpreted as carrying out a distributed primal-dual algorithm over the Internet to maximize aggregate utility, and a user's utility function is (often implicitly) defined by its TCP algorithm, see e.g. [4], [9], [11], [14], [10], [8], [5]. For TCP Reno (or its variants such as NewReno or SACK), this implies that the equilibrium source rate x_i and the overall dropping probability p_i satisfies

$$p_i \quad = \quad \frac{2}{2 + x_i^2(d+\tau)^2}, \quad i = 1, \dots, N \qquad (4)$$

This is equivalent to the well-known square-root p formula derived in, e.g., [6], [12], when the dropping probability p_i is small. At equilibrium, the flows share the bandwidth according to

$$x_i(1-p_i) \quad = \quad \mu_i c \qquad (5)$$

so that $\sum_i x_i(1-p_i) = c$.

Here, p_i is the probability that a packet from TCP flow i is dropped, either on arrival due to CHOKe and RED, or after it has been admitted into the queue. Even if a packet is not dropped on arrival, it can be dropped afterwards because a future arrival from the same flow triggers a comparison. To see why p_i is related to CHOKe and RED dropping probabilities according to (3), note that every arrival from flow i can trigger either 0 packet loss from the buffer, 1 packet loss due to RED, or 2 packet losses due to CHOKe. These events happen with respective probabilities of $(1-h_i)(1-r)$, $(1-h_i)r$, and h_i. Hence, each arrival to the buffer is accompanied by an average packet loss of

$$2h_i + (1-h_i)r + 0 \cdot (1 - h_i - (1-h_i)r)$$

Hence we take the overall loss probability p_i to be the packet loss rate $2h_i + (1-h_i)r$.

We next explain how to derive the queueing delay τ and the bandwidth share μ_i, the most subtle features of CHOKe.

C. *Spatial distribution and packet velocity*

If a packet cannot be dropped once it has been admitted into the queue, then, clearly, the queueing delay τ and the bandwidth share μ_i under FIFO are

$$\tau \;=\; \frac{b}{c} \quad \text{and} \quad \mu_i \;=\; \frac{b_i}{b} c \qquad (6)$$

For a "leaky queue" where a packet can be dropped while it advances towards the head of the queue, (6) no longer holds, and the queueing delay and bandwidth share depend critically on the spatial characteristics of the queue. The key to their understanding is the spatial distribution of packets in the queue and the "velocity" at which packets move through the queue at different positions, defined as follows. Let $y \in [0, b]$ denote a position in the queue, with $y = 0$ being the tail and $y = b$ the head of the queue. Let $\rho_i(y)$ be the probability that the packet at position y belongs to flow i, $i = 0, \dots, N$; as usual, we have $\rho_i(y) = \rho_1(y)$, $i \geq 1$, and

$$\rho_0(y) + \rho_1(y)N \quad = \quad 1, \quad \text{for all } y \in [0, b] \qquad (7)$$

The average number
of flow-i packets in the entire backlog satisfies

$$\int_0^b \rho_i(y)dy \quad = \quad b_i \qquad (8)$$

More importantly, the bandwidth share μ_i is the probability that the head of the queue is occupied by a packet from flow i:

$$\mu_i \quad = \quad \rho_i(b) \qquad (9)$$

Hence, to find μ_i, we need to solve for the spatial density $\rho(y)$; we will return to this after modelling another crucial feature of CHOKe, the velocity at which packets move through the queue.

We make two remarks before proceeding further. First, if the queue is not leaky, then the spatial distribution will be uniform, $\rho_i(y)$ being independent of position y. Then $\rho_i(y) = \mu_i$ for all $y \in [0, b]$ together with (8) implies the bandwidth share in (6). That is, the bandwidth share depends only on the total number of flow-i packets in the queue, and not on details of their distribution.

Moreover, it is exactly equal to the flow's share of backlog.

When the queue is leaky, however, (9) says that the bandwidth share of flow i depends on the spatial distribution of packets only at the *head* of the queue and does not depend directly on the distribution at other positions or the total number of flow-i packets, in stark contrast to the case of non-leaky queue. This is the underlying reason why UDP packets can occupy half of the buffer, yet receiving very small bandwidth share: when UDP rate is high, $\rho_0(y)$ decreases rapidly from $y = 0$ to $y = b$ with $\rho_0(b) \simeq 0$; see simulation results in Section III.

As we mention above, in a leaky queue, the queueing delay of a packet that eventually exits the queue is no longer the backlog it sees on arrival divided by the link capacity. This is because it advances towards the head both when packets in front of it exit the queue and when they are dropped by CHOKe. To model this dynamics, define $v(y)$ as the velocity at which the packet at position y moves towards the head of the queue:

$$v(y) \quad = \quad \frac{dy}{dt}$$

For instance, the velocity at the head of the queue equals the link capacity, $v(b) = c$. Then, the queueing delay τ is given in terms of $v(y)$ as

$$\tau \quad = \quad \int_0^\tau dt \quad = \quad \int_0^b \frac{1}{v(y)} dy \qquad (10)$$

We have completed the definition of spatial distribution $\rho_i(y)$ and velocity $v(t)$ of packets in the queue. We now derive an ordinary differential equation model of these quantities.

D. ODE model of $\rho_i(y)$ and $v(y)$

We will derive an ODE model for $\rho_0(y)$ and $v(y)$; $\rho_1(y)$ can then be obtained from (7).The key is to carefully model the dropping process after a packet has been admitted into the buffer.

Consider a small volume $v(y)dt$ of the (one-dimensional fluid) queue at position y. The amount of fluid (packets) in this volume that belongs to flow i is $\rho_i(y)v(y)dt$, $i = 0, 1$. For instance, $\rho_i(0)v(0)dt$, $i = 0, 1$, is the amount of fluid that arrives at the tail of the queue, packets that are not dropped by CHOKe or RED on arrival and admitted into the buffer. Hence

$$\rho_i(0)v(0) \quad = \quad x_i(1 - h_i)(1 - r), \quad i = 0, 1 \quad (11)$$

Another boundary condition is the packet velocity at the head of the queue mentioned above:

$$v(b) \quad = \quad c \qquad (12)$$

Suppose the small volume $\rho_i(0)v(0)dt$ of fluid (our "tagged packet") arrives at the buffer at time 0, and reaches position $y = y(t)$ at time t. During this period $[0, t]$, there are $x_i t$ packet arrivals from flow i, and each of these arrivals triggers a comparison. The tagged packet is selected for comparison with probability $1/b$ each time. We model this by saying that the fluid is thinned by a factor $(1 - 1/b)^{x_i t}$ when it reaches position y at time t. Thus

$$\rho_i(0)v(0)\left(1 - \frac{1}{b}\right)^{x_i t} \quad = \quad \rho_i(y)v(y)$$

Taking logarithm on both sides and using

$$t \quad = \quad \int_0^y \frac{1}{v(s)} ds$$

to eliminate t, we have

$$\ln(\rho_i(0)v(0)) + x_i \ln\left(1 - \frac{1}{b}\right) \int_0^y \frac{1}{v(s)} ds \quad = \quad \ln(\rho_i(y)v(y))$$

Differentiating both sides with respect to y, we get

$$\frac{x_0}{v(y)} \ln\left(1 - \frac{1}{b}\right) \quad = \quad \frac{\rho_0'(y)}{\rho_0(y)} + \frac{v'(y)}{v(y)} \qquad (13)$$

$$\frac{x_1}{v(y)} \ln\left(1 - \frac{1}{b}\right) \quad = \quad \frac{\rho_1'(y)}{\rho_1(y)} + \frac{v'(y)}{v(y)} \qquad (14)$$

(13) $\times\, \rho_0(y) +$ (14) $\times\, N \times \rho_1(y)$ then yields

$$v'(y) \quad = \quad (\rho_0(y)(x_0 - x_1) + x_1)\, \ln\left(1 - \frac{1}{b}\right) \quad (15)$$

where we have used (7). Substituting (15) into (13), we get

$$\rho_0'(y) = \ln\left(1 - \frac{1}{b}\right)\, (x_0 - x_1)\, \rho_0(y)(1 - \rho_0(y))\, \frac{1}{v(y)} \quad (16)$$

Hence the spatial distribution $\rho_i(y)$ and packet velocity $v(y)$ is given by the two-dimensional system of nonlinear differential equations (15)–(16) with boundary conditions (11)–(12).

E. Model summary and implications

To recapitulate, CHOKe is modelled by a mixed system of nonlinear algebraic and differential equations. The key equations are (4) that models the feedback loop between TCP flows and the dropping probability due to CHOKe and RED, (15) and (16) that model the spatial distribution and velocity of packets in the queue, with boundary conditions (11)–(12). Other quantities in these equations are specified in (1)–(3) on dropping probabilities, (5) and (9) on bandwidth sharing, (7) on $\rho_1(y)$, (8) on flow backlogs, and (10) on queueing delay. Our goal again is to understand the effect of UDP rate x_0 on bandwidth shares, given by (9), and queueing delay, given by (10). Although we cannot solve the model analytically, we can deduce several simple properties of CHOKe. These properties are accurately validated in the simulation results presented in Section III. The first property is the surprising feature of CHOKe that is contrary to a non-leaky FIFO queue.

Theorem 1.
For all x_0, $b_i \leq 1/2b$, $i = 0, 1$.
As $x_0 \to \infty$, $\mu_0 \to 0$.

Proof. 1. From (3), $2h_i + (1 - h_i)r = p_i \leq 1$ and hence using (2), we have

$$\frac{b_i}{b} = h_i = \frac{1+r}{2+r}$$

The right-hand side is a decreasing function for $r \geq 0$ with a maximum value of $1/2$ at $r = 0$.
2. From (16), we have

$$\frac{\rho_0'(y)}{\rho_0(y)(1-\rho_0(y))} = \beta(x_0 - x_1)\,\frac{1}{v(y)}$$

where $\beta := \ln(1 - 1/b)$. Define

$$\hat{\tau}(y) = \int_0^y \frac{dz}{v(z})$$

which can be interpreted as the time for a packet to reach position y from position 0. Integrating both sides from 0 to y, we get

$$\ln\frac{\rho_0(y)}{1-\rho(y)} - \ln\frac{\rho_0(0)}{1-\rho(0)} = \beta(x_0 - x_1)\hat{\tau}(y)$$

Hence,

$$\rho_0(y) = \frac{ae^{\beta(x_0-x_1)\hat{\tau}(y)}}{1 + ae^{\beta(x_0-x_1)\hat{\tau}(y)}} =: \frac{A(y;x_0)}{1 + A(y;x_0)}$$

for some constant $a > 0$. All other quantities in $A(y;x_0)$ except x_0 are bounded: $x_1 \leq c/N$ by symmetry, and given any $y > 0$, $\hat{\tau}(y)$ is bounded. Since $\beta < 0$, as $x_0 \to \infty$, the exponent tends to $-\infty$ and $A(y;x_0) \to 0$ for any $y > 0$. This implies in particular for $y = b > 0$ that $\mu_0 = \rho(b) \to 0$. □

In fact,the proof says something stronger: asymptotically as $x_0 \to \infty$, not only does $\mu_0 \to 0$, but all UDP packets are dropped near the tail of the queue by CHOKe! This is confirmed in the simulation results in the next section.

Some fine structures of the queueing process are also easy to derive. Generalizing (10), define $\tau(y)$ as the time for the packet in position y to exit the queue:

$$\tau(y) = \int_y^b \frac{dz}{v(z)}$$

Theorem 2. 1) *Packet velocity* $v(y)$ *is a convex and strictly decreasing function with* $v(0) = (x_0(1 - h_0) + x_1(1 - h_1)N)(1 - r)$ *and* $v(b) = 0$. *It is linear if and only if* $x_0 = x_1$.
2) *Queueing delay* $\tau(y)$ *is a strictly concave decreasing function.*
3) *The spatial distribution* $\rho_0(y)$ *of UDP packets is strictly increasing in* y *if* $x_0 < x_1$, *constant if* $x_0 = x_1$, *and strictly decreasing if* $x_0 > x_1$.

Proof. Using (7), (15) can be rewritten as

$$v'(y) = \beta(\rho_0(y)x_0 + \rho_1(y)x_1N) < 0$$

where $\beta = \ln(1 - 1/b) < 0$. Differentiating again and using $\rho'(y) + \rho'(y)N = 0$ from (7), we have

$$\begin{aligned} v''(y) &= \beta(\rho_0'(y)x_0 + \rho_1'(y)N) \\ &= \beta(x_0 - x_1)\rho_0'(y) \end{aligned}$$

From (16), we have

$$v''(y) = \beta^2(x_0 - x_1)^2\rho_0(y)\rho_1(y)\frac{N}{v(y)} \geq 0$$

with equality if and only if $x_0 = x_1$. The boundary values of $v(y)$ follows from (11) (sum over i) and (12).
2. It is easy to verify that $\tau'(y) < 0$ and $\tau''(y) < 0$ using $v(y)$ and $v'(y)$.
3. Follows from (16). □

We summarize these structural properties. First, the remaining queueing delay of a packet that eventually exit the queue decreases more than linearly (in space) as it progresses, due to the leaky nature of the buffer. Second, when x_0 is large, the spatial distribution $\rho_0(y)$ decreases rapidly towards the head of the queue. This means that most of the UDP packets are dropped before they reach the head. It is therefore possible to simultaneously maintain a large number of packets (concentrating near the tail) and receive a small bandwidth share, in stark contrast to the behavior of a non-leaky FIFO buffer. Indeed, as x_0 grows without bound, UDP share drops to 0.

F. Solution of CHOKe model

The mixed system of nonlinear algebraic and differential equations that model CHOKe is too complex to be solved exactly. In this subsection, we briefly describe how we solve it numerically. The numerical results are compared with ns-2 simulations in the next section.

For ease of reference, we reproduce below the system of differential equations with boundary conditions:

$$\begin{aligned}
\rho_0'(y) &= \ln\left(1-\frac{1}{b}\right)(x_0-x_1)\,\rho_0(y)(1-\rho_0(y))\,\frac{1}{v(y)} \\
v'(y) &= (\rho_0(y)(x_0-x_1)+x_1)\,\ln\left(1-\frac{1}{b}\right) \\
v(0) &= x_0(1-h_0)+Nx_1(1-h_1))(1-r) \\
\rho_0(0) &= x_0(1-h_0)(1-r)/v(0)
\end{aligned}$$

and the system of algebraic equations:

$$\begin{aligned}
p_i &= \frac{2}{2+x_i^2(d+\tau)^2}, \quad i=1,\ldots,N \\
v(b) &= c \\
r &= k(b-\underline{b}) \qquad \text{assuming } \underline{b}<b<\bar{b} \text{ in equilibrium} \\
h_i &= \frac{b_i}{b} \\
p_i &= 2h_i+(1-h_i)\,r \\
\mu_i c &= x_i(1-p_i) \\
\mu_i &= \rho_i(b) \\
1 &= \rho_0(y)+\rho_1(y)N, \quad \text{for all } y \\
\tau &= \int_0^b \frac{1}{v(y)}dy \\
b_i &= \int_0^b \rho_i(y)dy
\end{aligned}$$

To outline our numerical method, denote by $\alpha(y)$ the vector function:

$$\alpha(y) \;=\; (v(y),\rho_0(y)) \quad \text{for } y\in[0,b]$$

and by β the other variables:

$$\beta \;=\; (x_i,p_i,h_i,r,\mu_i,b_i,\tau,\rho_1,i=0,1)$$

Then the CHOKe model can be written into the following form:

$$\alpha'(y) \;=\; F(\alpha(y),\beta) \qquad (17)$$

$$\beta \;=\; G(\alpha(\cdot),\beta) \qquad (18)$$

with boundary conditions

$$f_1(\alpha(0)) \;=\; 0 \qquad (19)$$

Given any β, we can solve (17) and (19) numerically to obtain $\alpha(\beta) := \alpha(y;\beta)$, as a function of β. Substituting $\alpha(\beta)$ into (18), we obtain a fixed-point equation in β:

$$\beta \;=\; G(\alpha(\beta),\beta) \;=:\; g(\beta) \qquad (20)$$

A solution of our CHOKe model is a fixed point of (20). However, the straightforward fixed-point iteration of (20) does not converge to a solution of CHOKe.

Instead, we solved the nonlinear equation (20) approximately by minimizing the quadratic cost on the difference between β and $g(\beta)$:

$$\min_{\beta}\; J(\beta) \;:=\; (\beta-g(\beta))^T W(\beta-g(\beta))$$

with an appropriate choice of positive diagonal weighting matrix W. A solution β^* of CHOKe satisfies $J(\beta^*) = \min_\beta J(\beta) = 0$.

Matlab is used to implement above procedure. The weighting matrix is chosen such that each component in vector $W\beta$ is in range [10 100] nearby the fixed point. A direct search method (Nelder,Mead [16])for multidimensional unconstrained nonlinear minimization implemented in matlab, is used for this optimization problem. The search algorithm will be stopped when the $J(\beta)$ function get smaller than 0.05. This is accurate enough! The solutions are verified with the ns simulation in section III.

III. Simulation results

We implemented a CHOKe module in ns-2 version 2.1b9 and have conducted extensive simulations using the network shown in Figure 2 to study the equilibrium behavior of CHOKe. There is a single bottleneck link from router R1 to router R2 shared by N TCP sources and one UDP source. The UDP source sends data at constant rate (CBR). For all our simulations, the link capacity is fixed at c = 1Mbps and the round trip propagation delay is d = 100ms. We use RED+CHOKe as the queue management with RED parameters: (min_th $\underline{b} = 20$ packets, max_th $\bar{b} = 520$ packets, p_max = 0.5). Packet size is 1KB. The simulation time is 200–300 seconds.

We vary UDP sending rate x_0 from 12.5pkts/s to 1250 pkts/s and the number N of TCP flows from 12 to 64 to observe their effect on the equilibrium behavior of CHOKe. We measure, and compare with our numerical solutions, the following quantities

1) aggregate queue size b
2) UDP bandwidth share $\mu_0 = \rho_0(b)$
3) TCP throughput $x_1(1-p_1) = \mu_1 c = \rho_1(b)c$
4) spatial distribution $\rho_0(y)$ of UDP packets

The results illustrate both the equilibrium behavior of CHOKe and the accuracy of our analytical model. They show the ability of CHOKe to protect TCP flows and agree with those aggregate measurements of [17]. Moreover, they exhibit the fine structure of queue dynamics, and confirm the spatial character of our CHOKe model.

We next discuss these results in detail.

A. *Effect of UDP rate x_0*

First we study the effect of UDP sending rate on queue size and bandwidth allocation. The number of TCP sources is fixed at $N = 32$. We vary the UDP sending rate x_0 from 0.1$\times$ link capacity to 10$\times$ link capacity. The results are in Figure 3.

The aggregate queue length b steadily increases as UDP rate x_0 rises. UDP bandwidth share $\mu_0 = \rho_0(b)$ rises, peaks, and then drops to less than 5% as x_0 increases from $0.1c$ to $10c$, while the total TCP throughput follows an opposite trend eventually exceeding 95% of the capacity. We have shown both TCP throughput – the total number of TCP packets, from all N flows, transmitted by the link during each simulation – and TCP goodput – sum of largest sequence numbers in each simulation. TCP goodput does not count retransmitted packets and is hence lower than TCP throughput. These results match closely those obtained in [17], and with our analytical model.

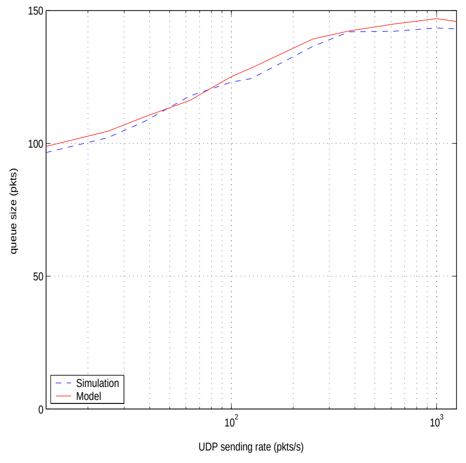
At first, one might expect that as UDP rate increases, the number of UDP packets in the queue steadily rises, and hence, under FIFO, UDP bandwidth share should steadily rise too, contrary to the result in Figure 3(b). This can be explained by the non-uniform spatial distribution of packets in the queue. These distributions are shown in Figure 4.

To get the packet distribution from each simulation (x_0 value), we took $J = 3000$ snapshots of the queue every 100ms for 300 seconds. From the J sample queue sizes b^j, we first calculated the average $b_{avg} := \sum_j b^j/J$. The distribution was estimated over this range $[0, b_{avg}]$, as follows. For each $y \in [0, b_{avg}]$, the sample distribution is calculated as

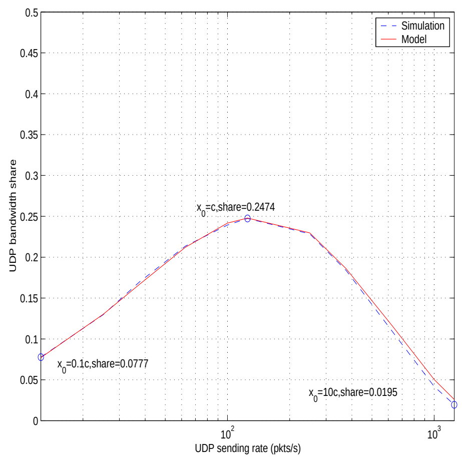
$$\rho(y) \quad = \quad \frac{1}{J}\sum_j \mathbf{1}_j(y)$$

where $\mathbf{1}_j(y)$ is 1 if the packet in position $\lfloor yb^j/b_{avg}\rfloor$ of the jth snapshot is UDP, and 0 otherwise. This distribution is plotted in Figure 4, together with the numerical solution of our CHOKe model.

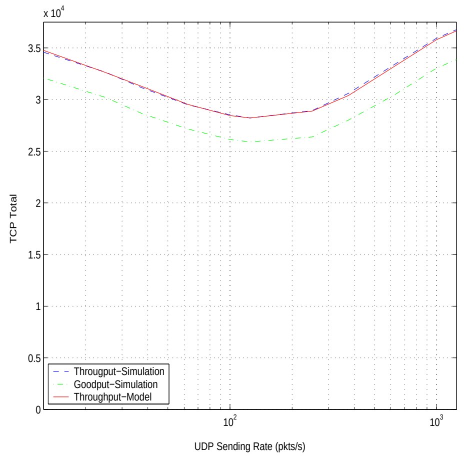
When $x_0 = 0.1c$ (Figure 4(a)), UDP packets are distributed roughly uniformly in the queue, with probability close to 0.08 at each position. As a result, its bandwidth share is roughly 10%. As x_0 increase, $\rho(y)$ becomes a decreasing function of y; moreover, the rate of decrease



(a) Queue size b

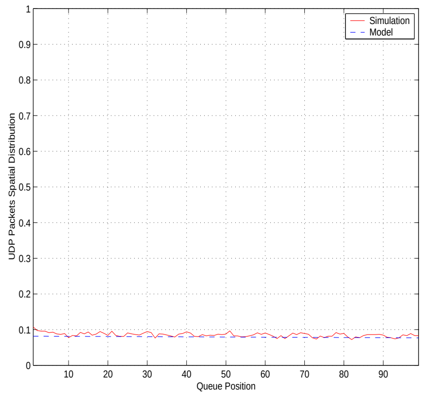


(b) UDP bandwidth share μ_0

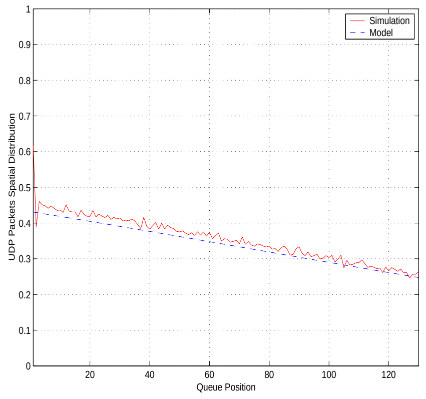


(c) Total TCP throughput $\mu_1 c$

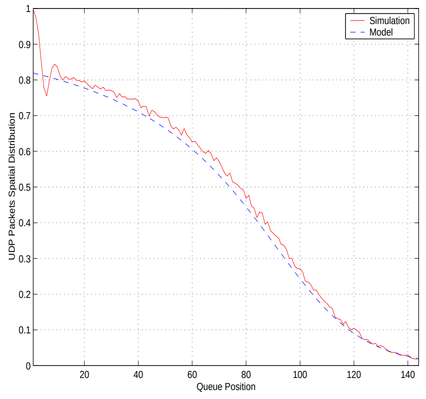
Fig. 3. Effect of UDP rate x_0 on queue size and bandwidth allocation. $N = 32$, $x_0 = 0.1c$ to $10c$, $c = 1$Mbps, simulation duration = 300sec.



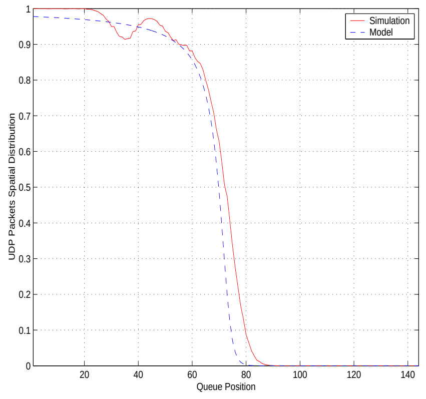
(a) UDP rate $x_0 = 0.1c$



(b) UDP rate $x_0 = c$



(c) UDP rate $x_0 = 10c$



(d) UDP rate $x_0 = 100c$

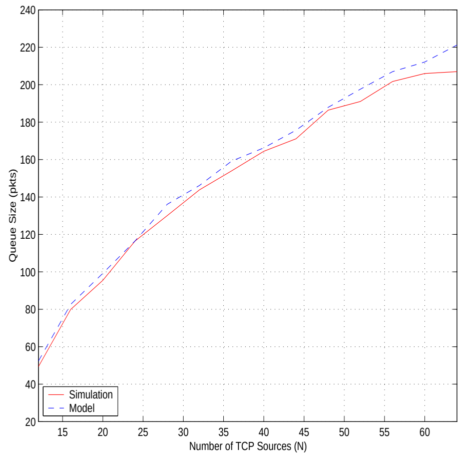
Fig. 4. Spatial distribution $\rho(y)$ of packets in queue at different UDP rates x_0. $N = 32$, $c = 125$pkts/s, simulation duration = 300sec.

rises. This can be understood from equation (16) that says that $\rho(y)$ is increasing in y when $x_0 < x_1$, uniform when $x_0 = x_1$, and decreasing when $x_0 > x_1$.

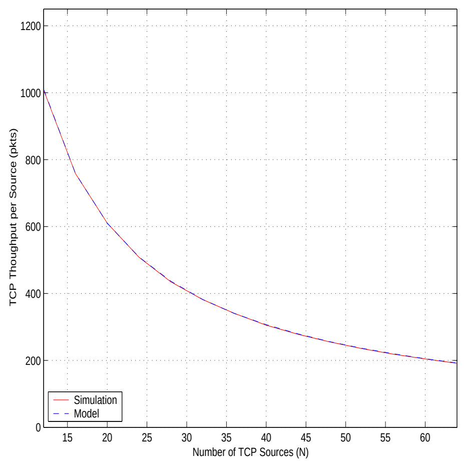
As the CHOKe model predicts, the UDP bandwidth share is determined only by the value of ρ at the head of the queue, $\mu_0 = \rho_0(b)$. Hence, as x_0 increases, μ_0 steadily drops to zero. Also marked in Figure 3(b), are the UDP bandwidth shares corresponding to UDP rates in Figure 4. As expected the UDP bandwidth shares in 3(b) are equal to $\rho(b)$ in Figure 4. When $x_0 > 10c$, even though roughly half of the queue is occupied by UDP packets, almost all of them are dropped before they reach the head of the queue!

B. *Effect of number N of TCP flows*

Figure 5 shows the effect of N on aggregate queue size b and on per-flow TCP throughput $\mu_1 c = \rho_1(b)c$.



(a) Queue size b



(b) TCP throughput per flow $\mu_1 c$

Fig. 5. Effect of number N of TCP flows on aggregate queue b and per-flow TCP throughput $\mu_1 c$. $N = 12 - 64$, $x_0 = 1250$ pkts/s, $c = 125$pkts/s, simulation duration = 300s

Not surprisingly, the queue size increases and per-flow

TCP throughput decreases with N as the queue becomes more congested. Again, the simulation and analytical results match very well, further validating our model.

IV. Additional Discussion on CHOKe

The model proposed above captures the equilibrium behavior of CHOKe very well under the assumption that the queue size b is always larger than $\underline{b}$. A sampled queue size can be found at figure(7). it is worth noting that this assumption may not hold for relatively small N.

With smaller N, each TCP source gets larger bandwidth share, and larger TCP sending rate requires lower dropping probability. However, when $b > \underline{b}$, the TCP dropping probability is at least $1/N$ due to CHOKe, because the UDP packets can take at most half of the queue. We can estimate the minimal N that will make CHOKe always active. From (3)

$$p_i = 2h_i + (1 - h_i)r \geq 2h_i \tag{21}$$

rewrite (4) here:

$$p_i \quad = \quad \frac{2}{2 + x_i^2(d+\tau)^2}, \quad i = 1, \ldots, N$$

Now

$$h_i \geq \frac{1}{2N} \tag{22}$$

When UDP sending rate is large, TCP take almost all the bandwidth, so

$$x_i \approx c/N \tag{23}$$

And around $\underline{b}$,

$$\tau \approx \underline{b}/c \tag{24}$$

In the example in last section, $N = 12$ can guarantee CHOKe to be always on, but $N = 8$ fails. Using the same parameters, we can estimate the minimal N to be 8.43. When $N = 8$, queue size will oscillate around $\underline{b}$,which means CHOKe turns on and off regularly as shown in figure (6). The model developed above cannot capture this situation.

For the similar reason, when N is fixed, and c increases linearly, the TCP source rate x_i will almost increase linearly. The dropping rate should decrease proportionally to $1/x_i^2$ based on equations (4,23) However, the dropping rate by CHOKe to TCP sources is at least $1/N$. This means CHOKe may drop too aggressive for the future high bandwidth network.

All the discussion above is about basic CHOKe whose flow chart is shown in figure1, In fact, the CHOKe can be implemented in different ways as mentioned in [17] for example: head CHOKe, tail CHOKe. They all have the same feature that the UDP packets will take almost half of the buffer when UDP's sending rate is high enough. But the bandwidth sharing feature can be totally different.

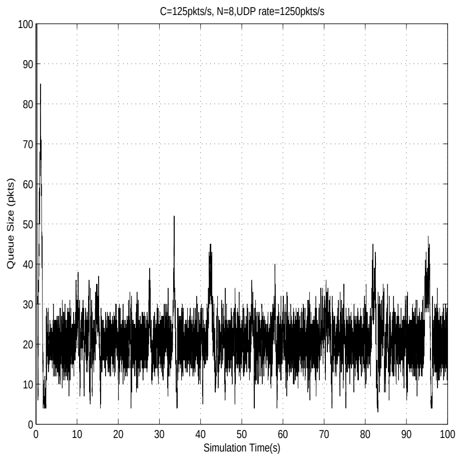


Fig. 6. Queue Size at N=8

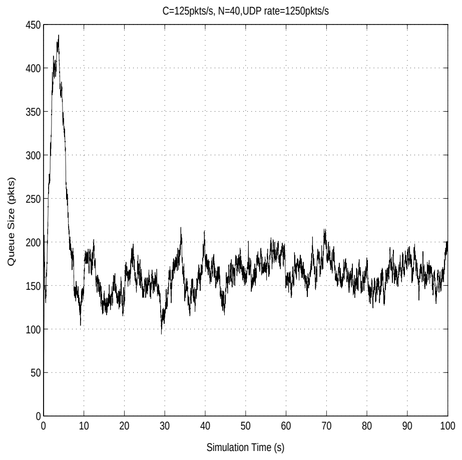


Fig. 7. Queue Size at N=40

This is another possible CHOKe implementation called Perfect CHOKe. Suppose we have perfect information about the queue. When the packets form ith flow comes,it will be dropped with probability $2b_i/b$. This implementation is easily modelled by a set of algebraic equations without the leaky buffer. More theoretical result can be derived based on this model later.

We showed UDP throughput in Figure 8. Unlike the basic CHOke, the UDP source can get almost half of the link bandwidth.

V. Conclusions

CHOKe is completely stateless and extremely simple to implement, yet surprisingly effective in bounding the bandwidth share of UDP flows. As shown in the simulations of [17], as UDP source rate increases, its bandwidth share eventually drops to zero, exactly opposite to what a

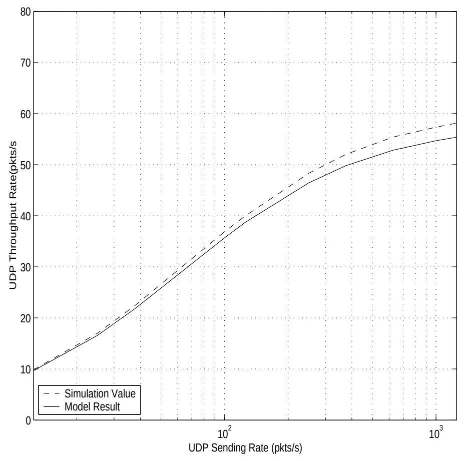


Fig. 8. UDP throughput rate (Perfect CHOKe)

regular FIFO queue behaves. To understand this precisely requires a careful modelling of the queueing process under CHOKe.

In this paper, we have derived a detailed model of CHOKe. Its key features are the incorporation of the feedback equilibrium of TCP detailed modelling of the queue dynamics. We have introduced the concepts of spatial distribution and velocity of packets in the queue, two quantities that are critical in understanding CHOKe. In a regular FIFO, both quantities are uniform across the queue. As a result, both the number of packets from a flow in the queue and its bandwidth share is proportional to its input rate. CHOKe however produces a "leaky buffer" where packets may be dropped as they move towards the head of the queue. This leads a non-uniform distribution of packets and their velocity, and makes it possible for a flow to simultaneously maintain a large number of packets in the queue and receive a vanishingly small bandwidth share. This is the main mechanism through which CHOKe protects TCP from UDP flows. As our model predicts, and simulation confirms, as UDP input rate increases, more and more UDP packets enter the queue. However, they concentrate near the tail of the queue, and are mostly dropped before they can advance to the head, resulting in small UDP throughput.

We also comment two features of CHOKe.One is that CHOKe algorithm may be turned on and off regularly when the link capacity is high or the number of TCP sources is small, which will make queue size oscillate around $\underline{b}$ The other is that different algorithms have different UDP throughput behaviors. We are still working to explore the details of these features.

REFERENCES

[1] Sanjeewa Athuraliya, Victor H. Li, Steven H. Low, and Qinghe Yin. REM: active queue management. *IEEE Network*, 15(3):48–53, May/June 2001. Extended version in *Proceedings of ITC17*, Salvador, Brazil, September 2001. `http://netlab.caltech.edu`.

[2] W. Feng, K G. Shin, D. Kandlur, and D. Saha. Stochastic Fair Blue: A queue management algorithm for enforcing fairness. In *Proceedings of INFOCOM*, April 2001.

[3] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397–413, August 1993. `ftp://ftp.ee.lbl.gov/papers/early.ps.gz`.

[4] Frank P. Kelly, Aman Maulloo, and David Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of Operations Research Society*, 49(3):237–252, March 1998.

[5] Srisankar Kunniyur and R. Srikant. End–to–end congestion control schemes: utility functions, random losses and ECN marks. In *Proceedings of IEEE Infocom*, March 2000. `http://www.ieee-infocom.org/2000/papers/401.ps`.

[6] T. V. Lakshman and Upamanyu Madhow. The performance of TCP/IP for networks with high bandwidth–delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, June 1997. `http://www.ece.ucsb.edu/Faculty/Madhow/Publications/ton97.ps`.

[7] Dong Lin and Robert Morris. Dynamics of random early detection. In *Proceedings of SIGCOMM'97*, pages 127–137, September 1997. `http://www.acm.org/sigcomm/sigcomm97/papers/p078.ps`.

[8] Steven H. Low. A duality model of TCP and queue management algorithms. In *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management (updated version)*, September 18-20 2000. `http://netlab.caltech.edu`.

[9] Steven H. Low and David E. Lapsley. Optimization flow control, I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, December 1999. `http://netlab.caltech.edu`.

[10] Steven H. Low, Larry Peterson, and Limin Wang. Understanding Vegas: a duality model. *J. of ACM*, 49(2):207–235, March 2002. `http://netlab.caltech.edu`.

[11] L. Massoulie and J. Roberts. Bandwidth sharing: objectives and algorithms. In *Infocom'99*, March 1999. `http://www.dmi.ens.fr/\%7Emistral/tcpworkshop.html`.

[12] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM Computer Communication Review*, 27(3), July 1997. `http://www.psc.edu/networking/papers/model_ccr97.ps`.

[13] P. McKenny. Stochastic fairness queueing. In *Proceedings of Infocom*, pages 733–740, 1990.

[14] Jeonghoon Mo and Jean Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5):556–567, October 2000.

[15] T. J. Ott, T. V. Lakshman, and L. Wong. SRED: Stabilized RED. In *Proceedings of IEEE Infocom'99*, March 1999. `ftp://ftp.bellcore.com/pub/tjo/SRED.ps`.

[16] Nelder, J.A. and Mead, R. A Simplex Method for Function Minimization In *Comput. J.*, pages 308-313, 1965

[17] Rong Pan, Balaji Prabhakar, and Konstantinos Psounis. CHOKe: a stateless AQM scheme for approximating fair bandwidth allocation. In *Proceedings of IEEE Infocom*, March 2000.

[18] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: achieving approximately fair bandwidth allocations in high speed networks. In *Proceedings of ACM Sigcomm*, 1998.