Maximum and Asymptotic UDP Throughput under CHOKe [*] [†]

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

ABSTRACT

A recently proposed active queue management, CHOKe, aims to protect TCP from UDP flows. Simulations have shown that as UDP rate increases, its bandwidth share initially rises but eventually drops. We derive an approximate model of CHOKe and show that, provided the number of TCP flows is large, the UDP bandwidth share peaks at $(e+1)^{-1} = 0.269$ when the UDP input rate is slightly larger than the link capacity, and drops to zero as UDP input rate tends to infinity, regardless of the TCP algorithm.

1. 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 is 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., [6, 2, 3, 10, 7, 9, 1], that aim at penalizing aggressive flows and ensuring fairness. The scheme, CHOKe, of [9] is particularly interesting in that it does not require any state information and yet can provide a minimum bandwidth share to TCP flows. The basic idea of CHOKe is explained in the following quote from [9]:

> 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 rates. In fact, as the arrival rate of UDP packets increases without bound, their bandwidth share approaches zero!

An intuitive explanation is provided in [9]: "the FIFO (first-in-first-out) 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. In [11], a detailed differential equation model of CHOKe is derived that clarifies the spatial characteristics of a leaky buffer. It is shown there that UDP packets are not uniformly distributed across the length of the queue. Instead, as UDP rate increases, even though the total number of UDP packets in the queue increase, the spatial distribution of UDP packets becomes more and more concentrated near the tail of the queue, and drops rapidly to zero toward the head of the queue. When a single UDP flow shares a link with N TCP flows, the number of UDP packets in the queue is almost N times that of a TCP flow, in the limit as UDP input rate tends to infinity. Despite their number, most of the UDP packets are 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. Though the model of [11] provides detail structural properties of TCP/CHOKe, it is too complex to solve analytically for the maximum throughput attainable by the UDP flow.

In this paper, we derive an approximate model of CHOKe and use it to derive the maximum and asymptotic UDP throughput. In Section 2 we present an algebraic model that allows us to numerically compute performance metrics, such as throughput, delay and loss probabilities. In Section

[*] **Proceedings of ACM Sigmetrics, San Diego, CA, June 2003.**
[†] We acknowledge the support of NSF through grants ANI-0113425 and ANI-0230967, and ARO through grant DAAD19-02-1-0283.

3, we prove analytically that, provided that the number of TCP flows is large, the maximum bandwidth share attainable by the UDP flow is $(e+1)^{-1} = 0.269$ and that this is attained when the UDP rate is $(2e-1)/(e+1) = 1.193$ times the link capacity (approximately, when the congestion based dropping probability is small). Moreover, it drops to zero as UDP rate increases without bound. The same results are also obtained in [8], independently using a different method.

These results are intrinsic to CHOKe and insensitive to the specific algorithms of TCP, the round trip delay, and congestion based dropping such as RED. We present ns simulations to validate these conclusions in Section 4.

2. CHOKE MODEL

The basic idea of CHOKe is summarized in Section 1. A flow chart describing its implementation with RED is given in Figure 2 from [9].

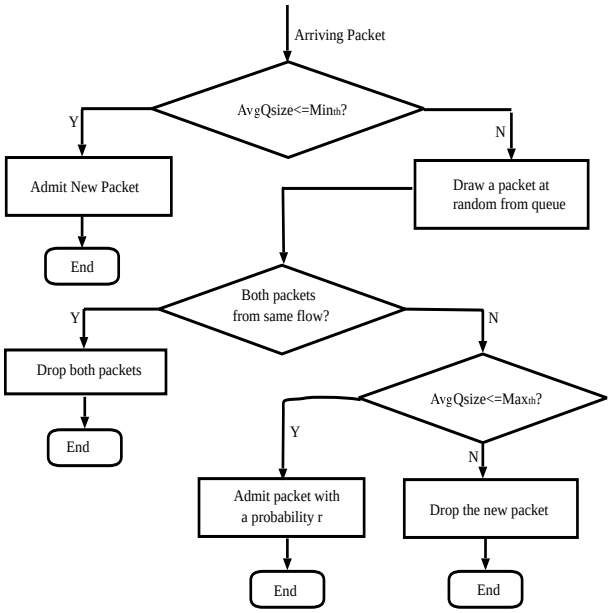


Figure 1: CHOKe flow chart from [8].

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 [9] that more drop candidate packets be used as the number of unresponsive flows increase. Here, we focus on the modeling of a single candidate drop 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 TCP flows and a single UDP flow. The TCP flows have round trip propagation delays $d_1, d_2, \ldots, d_N$ seconds respectively. We assume the system is stable and model its *equilibrium* behavior.

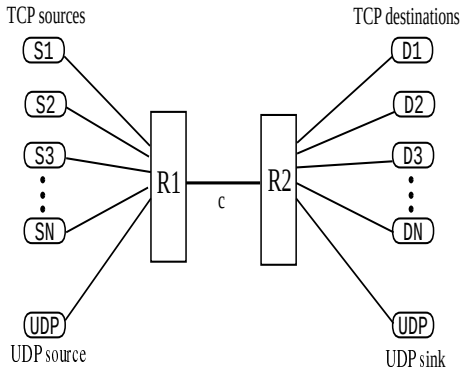


Figure 2: Network topology

2.1 Notations

Quantities (rate, backlog, dropping probability, etc) associated with the UDP flow are indexed by 0. Those associated with TCP flows are indexed by $1, \ldots, N$. These are equilibrium quantities which we assume exist.

Recall the setting $(c, N, d_i, i = 1, \ldots, N)$ where c is the link capacity, N is the number of TCP flows, and d_i, are the round-trip propagation delays of TCP flows. 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: total backlog, $b = \sum_{i=0}^{N} b_i$.

τ: common queueing delay. The round-trip delay for flow i is $d_i + \tau$.

r: Congestion-based dropping probability. Maximum and asymptotic UDP throughput are insensitive to the specific algorithm, such as RED, to compute this probability, as long as it is the same for all flows. In general, $r = g(b, \tau)$ for some function g as a function of aggregate backlog b and common queueing delay τ.

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

$$h_i = \frac{b_i}{b}$$

p_i: overall probability that packets of flow i, $i = 0, \ldots, N$, is dropped before it gets through, either by CHOKe or congestion based dropping:

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

The explanation of (1) is provided below.

x_i: source rate of flow i, $i = 0, \ldots, N$. Maximum and asymptotic UDP throughput are insensitive to the specific TCP algorithm, such as Reno or Vegas. In general, $x_i = f_i(p_i, \tau)$ for some function f_i as a function of overall loss probability p_i and queueing delay τ at equilibrium, $i = 1, \ldots, N$

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 notations. 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 congestion based dropping) is $x_i(1-h_i)(1-r)$, and the rate at which flow i packets exit the queue (throughput) is $x_i(1-p_i)$. Clearly, $x_i(1-p_i) \le x_i(1-h_i)(1-r) \le x_i$.

2.2 Model

We make three key approximations in our model. The first approximation, described next, yields an algebraic model that allows us to numerically compute performance metrics such as throughput, loss probabilities, and queueing delay. The other two approximations, described in the next section, lead to a simple method to estimate the maximum and asymptotic UDP throughput under CHOKe.

A packet may be dropped, either on arrival due to congestion (e.g., according to RED) or CHOKe, or after it has been admitted into the queue when a future arrival from the same flow triggers a comparison. We approximate the system by one in which the order of congestion based dropping and CHOKe is reversed: a packet is admitted with probability $1-r$, and if it is admitted, it is then compared with a packet randomly chosen from the queue and dropped with probability h_i.

With this approximation, the probability that a packet from flow i is eventually dropped is given by (1). To see this, note that every arrival from flow i can trigger either 0 packet loss, 1 packet loss due to congestion, or 2 packet losses due to CHOKe. These events happen with respective probabilities of $(1-h_i)(1-r)$, r, and $(1-r)h_i$. Hence, each arrival to the buffer is accompanied by an average packet loss of

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

and hence the overall loss probability p_i in (1).[1] This implies that the probability that a packet of flow i goes through the queue without being dropped is:

$$1 - p_i \quad = \quad (1-r)(1-2h_i) \tag{2}$$

We now derive this probability from another perspective.

Consider a packet of flow i that eventually goes through the queue without being dropped. The probability that it is not dropped on arrival is $(1-r)(1-h_i)$. Once it enters the queue, it takes τ time to go through it. In this time period, there are on average $\tau x_i(1-r)$ packets from flow i that survive congestion based dropping and arrive at the queue. The probability that this packet is not chosen for comparison is

$$\left(1 - \frac{1}{b}\right)^{\tau x_i(1-r)}$$

Hence, the overall probability that a packet of flow i survives the queue is

$$1 - p_i \quad = \quad (1-r)(1-h_i)\left(1 - \frac{1}{b}\right)^{\tau x_i(1-r)} \tag{3}$$

We will equate p_i in (2) and (3) to derive an equation which is one of the (two) key equations that allow us to estimate the maximum and asymptotic UDP throughput (see Section 3).

A simple interpretation of a leaky buffer is as follows: x_i is the source rate of flow i and $x_i(1-r)(1-h_i)$ is the rate at which flow i enters the queue after congestion-based dropping and CHOKe. This flow splits into two flows: one eventually exits the queue and the other is dropped inside the queue by CHOKe. The rate of the former flow is flow i's *throughput* $x_i(1-p_i) = x_i(1-r)(1-2h_i)$ and the rate of the latter flow is its *leak rate* $x_i(1-r)h_i$, so that they sum to the input rate $x_i(1-r)(1-h_i)$. Since the link is fully utilized, the flow throughput sum to link capacity:

$$x_0(1-p_0) + \sum_{i=1}^{N} x_i(1-p_i) \quad = \quad c \tag{4}$$

This completes the description of the model. In summary, the independent variable is UDP rate x_0. The dependent variables of the model are:

- backlogs b_i of flow i, $i = 0, \ldots, N$; total backlog $b = \sum_{i=0}^{N} b_i$.
- congestion based dropping probability r, CHOKe dropping probabilities h_i, and overall dropping probabilities p_i, $i = 0, \ldots, N$.
- TCP rate x_i, $i = 1, \ldots, N$.
- queueing delay τ.

The relations among these variables define our model. For ease of reference, we reproduce these ten equations here:

$$1 - p_i \quad = \quad (1-r)(1-2h_i), \qquad i = 0, \ldots, N \tag{5}$$

$$1 - p_i \quad = \quad (1-r)(1-h_i)\left(1 - \frac{1}{b}\right)^{\tau x_i(1-r)} \tag{6}$$

$$h_i \quad = \quad \frac{b_i}{b}, \qquad i = 0, \ldots, N \tag{7}$$

$$c \quad = \quad x_0(1-p_0) + \sum_{i=1}^{N} x_i(1-p_i) \tag{8}$$

$$b \quad = \quad \sum_{i=0}^{N} b_i \tag{9}$$

$$x_i \quad = \quad f_i(p_i, \tau), \qquad i = 1, \ldots, N \quad \text{(TCP)} \tag{10}$$

$$r \quad = \quad g(b, \tau) \qquad \text{(e.g. RED)} \tag{11}$$

We assume that this set of equations has a solution for $(x_i, \tau, r, b, b_0, b_i, h_0, h_i, p_0, p_i, i = 1, \ldots, N)$, once the TCP

[1] The overall loss probability for the original CHOKe algorithm is (see [11]):

$$p_i \quad = \quad 2h_i + r - rh_i$$

which is larger than the approximate loss probability given by (1) because a packet that is first dropped due to congestion saves a potential loss of two packets due to CHOKe. The difference however is small since both r and h_i are typically small.

algorithm $x_i = f_i(p_i, \tau)$ and AQM algorithm $r = g(b, \tau)$ are specified, and that the limits of these quantities as $x_0 \to \infty$ exist.

Our goal is to compute the maximum and asymptotic values (as $x_0 \to \infty$) of UDP throughput $x_0(1-p_0)$. We now show that if N is large, then only three equations are needed for this calculation (the two equations (5)–(6) for $i = 0$ and a new equation (13) below that replaces (5)–(7) for $i = 1, \ldots, N$). It suggests that these are intrinsic properties of CHOKe and are insensitive to TCP/AQM algorithms (f_i, g).

3. THROUGHPUT ANALYSIS

The second key assumption we make is that N is large. Since

$$h_i \;=\; \frac{b_i}{b_0 + \sum_{j=1}^N b_j}$$

we assume for TCP sources that h_i is on the order of $1/N$, so that $h_i \simeq 0$ when N is large, for $i = 1, \ldots, N$. Then a comparison triggered by a *TCP* packet arrival seldom yields a match. This means that, once in the queue (after congestion based dropping), a TCP packet will never be dropped. The overall dropping probability p_i then reduces to (substitute $h_i = 0$ into (1)):

$$p_i \;=\; r, \quad i = 1, \ldots, N$$

More importantly, this provides a simple relation between queueing delay, throughput and backlog. Since there is no "leaking" for TCP packets in the buffer, the throughput rate for TCP flow i is $x_i(1-p_i), i = 1, \ldots, N$. There are b_i packets in the buffer from this TCP flow i. Then Little's Theorem implies:

$$\tau \;=\; \frac{b_i}{x_i(1-p_i)} \quad \text{for} \quad i = 1, \ldots, N \tag{12}$$

From the condition (8) of full link utilization, the aggregate TCP throughput is $\sum_{i=1}^N x_i(1-p_i) = c - x_0(1-p_0)$. The aggregate number of TCP packets in the buffer is $\sum_{i=1}^N b_i = b - b_0 = b(1-h_0)$. Summing the numerators and denominators on the right side of (12) over $i = 1, \ldots, N$, we get:

$$\tau \;=\; \frac{\sum_{i=1}^N b_i}{\sum_{i=1}^N x_i(1-p_i)} = \;\frac{b(1-h_0)}{c - x_0(1-p_0)} \tag{13}$$

This assumption makes the estimate of UDP throughput insensitive to TCP algorithms.

Note that with this assumption, the model (5)–(11) is simplified to dependent variables $(x_i, \tau, r, b, b_0, b_i, h_0, p_0,\ i = 1, \ldots, N)$, with $h_i = 0$ and $p_i = r$, $i = 1, \ldots, N$, and equations with the three equations (5)–(7) for $i = 1, \ldots, N$ replaced by the single equation (13).

The third key approximation we make is that the total backlog b is large so that

$$\left(1 - \frac{1}{b}\right)^b \;\simeq\; e^{-1}$$

Equating $1 - p_i$ in (5) and (6) for $i = 0$, we have

$$\left(1 - \frac{1}{b}\right)^{\tau x_0(1-r)} \;=\; \frac{1-2h_0}{1-h_0} \tag{14}$$

Substituting (13) into (14) to eliminate τ, and apply the third approximation, we have

$$\frac{1-h_0}{1-2h_0} \;=\; \exp\left(\frac{x_0(1-r)(1-h_0)}{c - x_0(1-r)(1-2h_0)}\right) \tag{15}$$

where we have used (5). This equation yields the maximum and asymptotic UDP throughput.

THEOREM 1. *1. The maximum UDP bandwidth share is $\mu_0^* = (e+1)^{-1} = 0.269$.*

2. It is attained when the UDP input rate after congestion based dropping is $x_0^*(1-r^*) = c(2e-1)/(e+1) = 1.193c$.

3. In this case, the CHOKe dropping rate for UDP is $h_0^* = (e-1)/(2e-1) = 0.387$.

Proof. Denote UDP bandwidth share by (using (5))

$$\mu_0 \;:=\; (1-p_0)\frac{x_0}{c} \;=\; (1-r)(1-2h_0)\frac{x_0}{c}$$

Then rewrite (15) as

$$\frac{1-h_0}{1-2h_0} \;=\; \exp\left(\frac{1-h_0}{1-2h_0} \cdot \frac{\mu_0}{1-\mu_0}\right) \tag{16}$$

Let $\gamma(h_0)$ denote

$$\gamma(h_0) \;:=\; \frac{1-h_0}{1-2h_0} \tag{17}$$

Then (16) becomes:

$$\gamma(h_0) \;=\; e^{\gamma(h_0)\frac{\mu_0}{1-\mu_0}}$$

or

$$\mu_0 \;=\; \frac{\ln \gamma(h_0)}{\gamma(h_0) + \ln \gamma(h_0)} \tag{18}$$

It is easy to check that the right-hand side has a unique maximum at $\gamma(h_0) = e$ with maximum bandwidth share μ_0^* given by

$$\mu_0 \;\le\; \max_\gamma \frac{\ln \gamma}{\gamma + \ln \gamma} \;=\; \frac{1}{1+e} \;=:\; \mu_0^* \tag{19}$$

Substituting $\gamma(h_0) = e$ into the definition (17) of $\gamma(h_0)$, the maximum UDP bandwidth share is attained when the CHOKe dropping probability h_0 for UDP is

$$h_0^* \;=\; \frac{e-1}{2e-1} \tag{20}$$

Since $h_0 = b_0/b$, this implies that 39% of the queue are UDP packets when UDP attains the highest throughput.

Since $\mu_0^* = x_0^*(1-r^*)(1-2h_0^*)/c$, the UDP rate after congestion based dropping, $x_0^*(1-r^*)$, that attains the maximum throughput is

$$x_0^*(1-r^*) \;=\; \frac{\mu_0^* c}{1-2h_0^*}$$

Substituting (19) and (20), we have

$$x_0^*(1-r^*) \quad = \quad \frac{2e-1}{e+1}c$$

□

The next result says that the UDP throughput $x_0(1-p_0)$ drops to zero as UDP rate x_0 grows without bound, under the assumption that rates x_i under TCP algorithm remains finite when loss probability p_i becomes large, i.e., $\lim_{p_i \to 1} f_i(p_i, \tau) < \infty$, $i = 1, \ldots, N$. This assumption is always satisfied when TCP flows are controlled not to send packets infinitely fast when the capacity is fixed.

THEOREM 2. *Suppose* $\lim_{p_i \to 1} f_i(p_i, \tau) < \infty$, $i = 1, \ldots, N$. *As* $x_0 \to \infty$, $b_0 \to b/2$ *but* $\mu_0 \to 0$.

Proof. Note that from (5),

$$h_0 \quad = \quad \frac{1}{2} \cdot \frac{p_0 - r}{1-r} \quad \leq \quad \frac{1}{2}$$

since $p_0 \leq 1$. We argue that $b_0/b = h_0 \to 1/2$ as $x_0 \to \infty$. From (8), we have $x_0(1-p_0) \leq c$ for all x_0. Hence $p_0 \to 1$ as $x_0 \to \infty$. Hence from (5), we have

$$(1 - r(\infty))(1 - 2h_0(\infty)) \quad = \quad 0$$

where (all limits exist by assumption)

$$r(\infty) \quad := \quad \lim_{x_0 \to \infty} r \qquad \text{and} \qquad h_0(\infty) \quad := \quad \lim_{x_0 \to \infty} h_0$$

Hence either $r(\infty) = 1$ or $h_0(\infty) = 1/2$. If $r(\infty) = 1$, then $\lim_{x_0 \to \infty} p_i = r(\infty) = 1, \quad i = 1, \ldots, N$. This implies that $x_i = f_i(p_i, \tau) < \infty$ as $x_0 \to \infty$. The condition of full link utilization (8) then implies that UDP throughput $x_0(1-p_0) = c$. This violates Theorem 1. Hence $r(\infty) < 1$ and $h_0(\infty) = 1/2$.

Then $\gamma \to \infty$ from (17), and hence, using (18), $\mu_0 \to 0$. □

We visualize equation (15) in Figure 3. It illustrates both theorems above.

We close this section by presenting another way to derive the key equation (14). The rate of flow i is $x_i(1-r)(1-h_i)$ when it first enters the tail of the queue after congestion-based dropping and CHOKe, and it takes τ seconds for packets to reach the head of the queue. After traveling down the queue for t seconds, $0 \leq t \leq \tau$, the packets arrive at a certain point $y(t) \in [0, b]$, where it has been thinned by a factor $(1-1/b)^{x_i(1-r)t}$ (following the same argument that leads to (3)) and the rate of the flow at $y(t)$ is

$$\tilde{x}_i(t) \quad := \quad x_i(1-r)(1-h_i)\left(1 - \frac{1}{b}\right)^{x_i(1-r)t}$$

Hence $\tilde{x}_i(t)dt$ is the infinitesimal volume of the fluid at the point $y(t)$ in the queue, and the backlog from flow i is thus

$$b_i \quad = \quad \int_0^\tau x_i(1-r)(1-h_i)\left(1 - \frac{1}{b}\right)^{x_i(1-r)t} dt$$

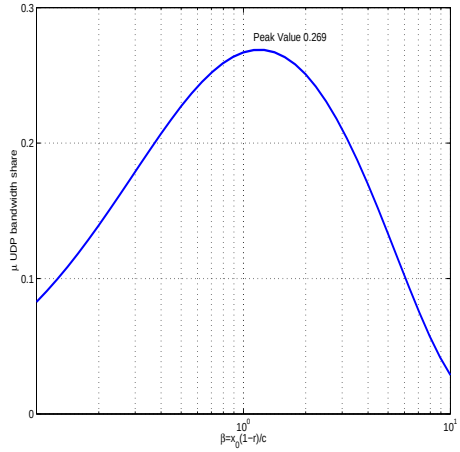


Figure 3: μ_0 **v.s.** $x_0(1-r)/c$

If we approximate $(1-1/b)^b$ by e^{-b}, when b is large, then the above integral reduces to

$$\frac{b_i}{b} \quad = \quad (1-h_i)\left(1 - e^{-x_i(1-r)\tau/b}\right)$$

In particular, since $h_0 = b_0/b$, this implies

$$\frac{1-2h_0}{1-h_0} \quad = \quad e^{-x_0(1-r)\tau/b}$$

This is equivalent to (14) when we approximate $(1-1/b)^b$ by e^{-1}.

4. SIMULATIONS

In this section, we present simulation results to validate our model (5)–(11) and throughput analysis. We implemented a CHOKe module with RED in ns-2 version 2.1b9.

4.1 TCP/AQM algorithms

The model for RED dropping probability is

$$r \quad = \quad k(b - \underline{b})$$

where $k := p_{max}/(\overline{b} - \underline{b})$ is the slope of the RED drop profile. We chose parameters of our simulations so that the equilibrium queue length b always lied between the minimum queue threshold $\underline{b}$ and the maximum queue threshold $\overline{b}$.

We have conducted simulations with TCP NewReno,TCP Vegas and a modified Vegas to verify that the UDP throughput is insensitive to TCP algorithms. TCP Reno (or its variants such as NewReno or SACK) is modeled by the following relation between the equilibrium source rate x_i, the overall dropping probability p_i, and the round trip time $d_i + \tau$ (see e.g., [4]):

$$p_i \quad = \quad \frac{2}{2 + x_i^2(d_i + \tau)^2}$$

This is equivalent to the well-known square-root-p formula when the loss probability is small.

For TCP Vegas, the source rate x_i is related to the round trip propagation delay d_i and queueing delay τ in equilibrium

according to (see [5]):

$$x_i = \frac{\alpha d_i}{\tau} \quad (21)$$

where α is a protocol parameter. In equilibrium, the ith Vegas source puts αd_i number of packets in the queue and hence the total number of TCP packets in the queue is $\sum_{i=1}^{N} \alpha d_i$. Under the large-$N$ assumption where TCP packets are not leaked from the queue, the queueing delay, given by (13), is then

$$\tau = \frac{\sum_{i=1}^{N} \alpha d_i}{c - x_0(1-p_0)} \quad (22)$$

Since $x_0(1-p_0) = 0$ both when $x_0 = 0$ or when $x_0 = \infty$ (Theorem 2), queueing delay under TCP Vegas is small both when UDP rate is small and when it is large. This interesting prediction is verified by the simulation below.

Note that the model (21)–(22) of Vegas assumes a Droptail router that has a large enough buffer capacity to accommodate the $\sum_{i=1}^{N} \alpha d_i$ packets that Vegas sources attempt to keep in the network, so that there are no packet loss in equilibrium; see [5]. RED and CHOKe dropping violates this assumption. As a result, Vegas reacts more or less like Reno: halving its window on each drop and increases it by one packet per round trip time.

We present simulation results with three TCP implementations, NewReno, Vegas, and modified Vegas. In modified Vegas, the source does not halve its sending window when there is a loss. In all our simulations large buffer capacity is used so that all packet drops are due to RED or CHOKe. Since modified Vegas uses only queueing delay as congestion feedback, the equilibrium behavior of modified Vegas matches well the model (21)–(22). We will see that the throughput share μ_0 as a function of UDP rate $x_0(1-r)/c$ is insensitive to these TCP implementations.

Our simulations focus on TCP Vegas both to show that UDP throughput is insensitive to TCP algorithm and because TCP Vegas scales better than TCP Reno with respect to link capacity, especially under CHOKe. This is because CHOKe increases the overall loss probability, limiting the achievable rate of TCP Reno. Since TCP Vegas sets its rate based on queueing delay, it does not have this limitation.

4.2 Simulation setup

We simulated the network in Figure 2 to study the equilibrium behavior of CHOKe. There is a single bottleneck link from router R1 to router R2 shared by $N = 100$ TCP sources and one UDP source. The UDP source sends data at a constant rate. The link capacity is fixed at $c = 15$Mbps for all simulations. We use RED+CHOKe as the queue management with RED parameters: (min_th $\underline{b} = 20$ packets, max_th $\overline{b} = 1020$ packets, $p_{max} = 0.1$). Packet size is 1KBytes. The simulation time is 20 seconds.

The original Vegas implementation in ns-2 works poorly in a lossy environment, for two reasons. First the implementation estimates RTT naively by setting it to the difference between sending time of a packet and receiving time of its ACK. When packets are lost, the ACK may be a duplicate ACK or it may be triggered by the retransmitted packet. A more sophisticated estimation is required when losses are frequent. Second, when there are multiple losses in the same round trip time, which is not infrequent since CHOKe drops two packets every time a comparison yields a match, the Vegas implementation often incurs timeout and slow-start.

As a consequence, we implemented a modified Vegas module in ns-2 based on the NewReno code which better handles losses. There are three major changes. First, we do not estimate RTT with duplicate ACKs, especially with the first and second duplicate ACKs when fast retransmit/fast recover phase is not yet entered. Second we use the NewReno's fast retransmit and fast recovery code to deal with multiple losses. Third, we change the Vegas code such that it only response to queueing delay not loss. These changes should not affect the equilibrium behavior of the TCP algorithms.

In our simulations, we vary UDP sending rate x_0 from $0.1c = 1.5$Mbps to $10c = 150$Mbps. We measure the following quantities, and compare with the numerical solutions of (5)–(11):

1. Normalized UDP bandwidth share $\mu_0 = x_0(1-p_0)/c$.
2. aggregate queue size b
3. round-trip time $d_i + \tau$

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 [9]. We next discuss these results in detail.

4.3 Simulation results

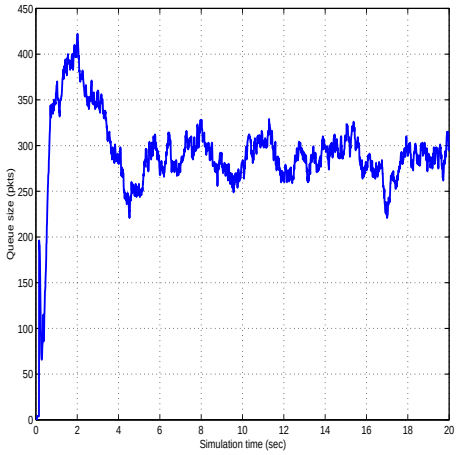
We show two sets of results. The first set presents measurements of buffer size, queueing delay and throughput share using the modified Vegas with N identical flows. It validates our model. The second set validates Theorems 1 and 2, and confirms that UDP share μ_0 is insensitive to TCP algorithms, RED parameters, and round trip delay.

4.3.1 Equilibrium properties

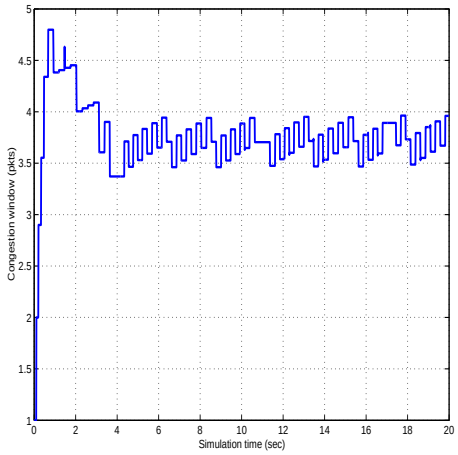
Since extensive simulation results have been previously reported in [9] and [11] with TCP Reno, we focus here on modified Vegas which matches model (21)–(22) well.

In the simulations of modified Vegas in this subsection, a common round trip delay $d = 100ms$ is used. We set α such that each source tries to put $\alpha d = 2$ packets in the buffer. The system is stable and converges to a steady state (modulo random fluctuations) very quickly. A sample queue size and congestion window are given in Figure 4, for UDP rate $x_0 = 15$Mbps, one of the data points in Figure 5.

Figure 5 illustrates the effect of UDP rate x_0 on queue size and queueing delay under modified Vegas. Each of the figures shows three curves: measurements from ns-2 simulations, numerical solution of the complete model (5)–(11), and that based on the simplified model described at the beginning of Section 3. The difference between the numerical solutions of the complete model and that of the simplified

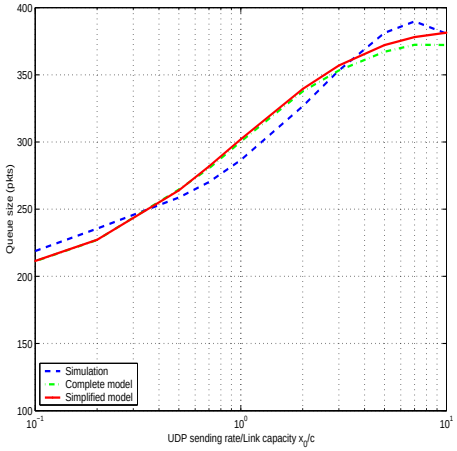


(a) Queue size v.s. time

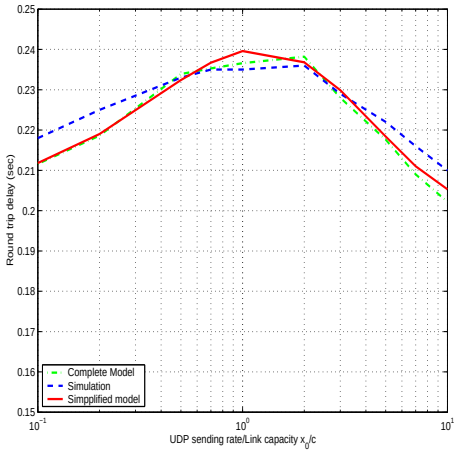


(b) Congestion window v.s. time

Figure 4: Example queue size and congestion window under modified Vegas



(a) Queue size b



(b) Round trip time $\tau + d$

Figure 5: Effect of UDP rate x_0 on queue size and queuing delay under modified Vegas. $N = 100$, $d = 100$ms, $c = 15$Mbps, $x_0 = 0.1c$ to $10c$, simulation duration = 20sec.

model is negligible. The error between simulation results and the numerical solutions is always less than 5 percent.

When UDP rate x_0 is small, the queue size is close to $N\alpha d =$ 200pkts. The aggregate queue length b steadily increases as UDP rate x_0 rises. It will approach 400pkts if there were no RED dropping because when UDP rate is very large, UDP packets take up half of the queue (see proof of Theorem 2) and there are about $N\alpha d = 200$ Vegas packets in the buffer ($N = 100$ is large). With RED dropping, the value is slightly less than 400pkts. As discussed in Section 4.1, the queueing delay is small both when $x_0 = 0.1c$ and when $x_0 = 10c$.

The UDP bandwidth share is shown in Figure 6. Similarly there are three curves from ns-simulation, complete model, and simplified model. The curve corresponding to the simplified model is marked as "Intrinsic UDP bandwidth share", and is the same curve in Figure 3, because its calculation does not depend on specific TCP algorithm f_i, or AQM algorithm g, or delay values. It is calculated using equation (15). The complete model matches the simulation better, but the prediction from the intrinsic curve is also acceptable.

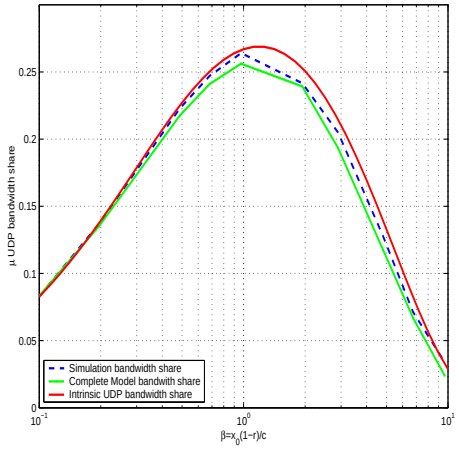
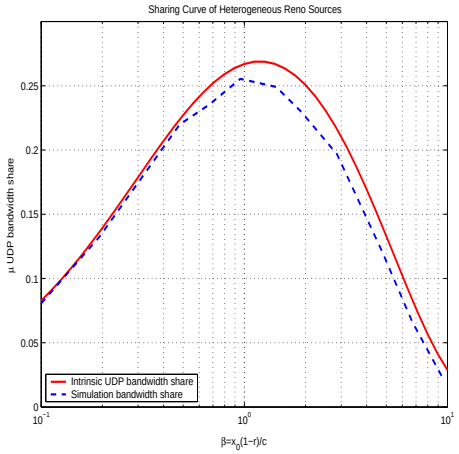


Figure 6: UDP bandwidth share μ_0 v.s. UDP rate after RED dropping $x_0(1-r)/c$

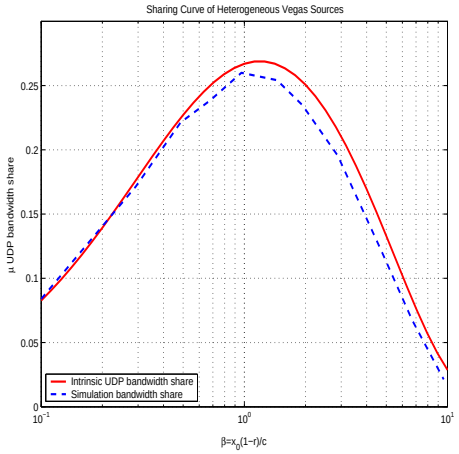
4.3.2 *Throughput share*

The second set of results, shown in Figure 7, illustrates the UDP bandwidth share for three TCP implementations: NewReno, modified Vegas, and original Vegas. Each subfigure has two curves: from simulation, and from the simplified model. The curves from the model are the same curve in Figure 3, and hence different TCP implementations yield similar simulation curves. The 100 sources have various propagation delays with 10 sources at each of the $50, 60, \ldots, 140$ms delay values. Figure 8 shows the corresponding curves for the original Vegas when all sources have the same delay of 100ms. Comparing Figures 6, 7, and 8, we see that the throughput share is sensitive neither to TCP implementation nor propagation delay. Theorems 1 and 2 predict that the UDP bandwidth share peaks at around 0.269 and tends to zero as x_0 increases. Simulation shows a slightly smaller peak UDP share. The discrepancy is probably due to the approximations we made in our model.

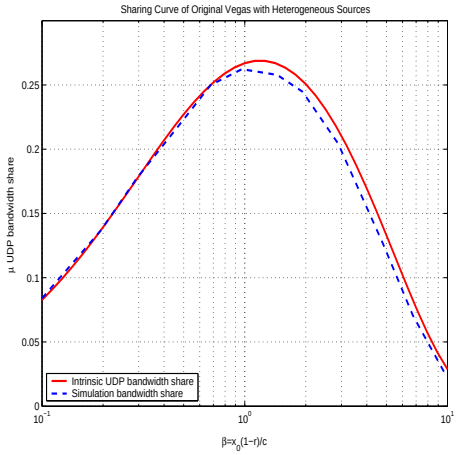
Finally, the UDP bandwidth share with a different slope of



(a) NewReno: $d_i = 50, 60, \ldots, 140$ms

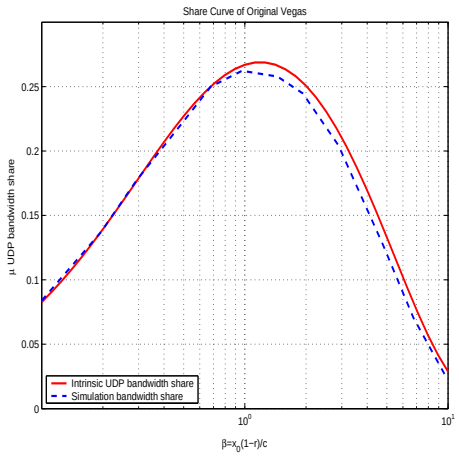


(b) Modified Vegas: $d_i = 50, 60, \ldots, 140$ms



(c) Original Vegas: $d_i = 50, 60, \ldots, 140$ms

Figure 7: UDP bandwidth share μ_0 as a function of UDP rate after RED dropping $x_0(1-r)/c$. $N = 100$, $c = 15$Mbps.



(a) Original Vegas: $d_i = 100$ms

Figure 8: UDP bandwidth share μ_0: Original Vegas, $d_i = 100$ms for all sources, $N = 100$, $c = 15$Mbps.

RED's drop profile is shown in Figure 9. The RED parameters are : (min_th $\underline{b} = 20$ packets, max_th $\overline{b} = 420$ packets, $p_{max} = 0.1$). Here we have modified $\overline{b}$ from 1020 to 402 to get a steeper slope of the drop profile than used in earlier simulations. The sources are NewReno with the same delay values as the previous set of simulations. It shows that the bandwidth share is almost independent of RED parameters.

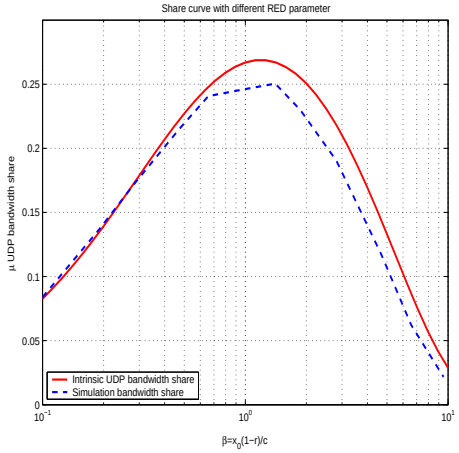


Figure 9: UDP bandwidth share μ_0: different RED configuration, NewReno.

5. CONCLUSION

We have developed an analytical model of CHOKe which is not only much simpler to solve numerically than the previous model of [11], but also allows us to compute analytically the maximum and asymptotic throughput. In particular, we prove that UDP bandwidth share peaks at $(e+1)^{-1} = 0.269$ when UDP rate is slightly larger than link capacity, and drops to zero when UDP rate approaches infinity.

The models and results here and in [11] complement each other. Here, we compute the macroscopic properties of CHOKe. The differential equation model of [11] provides detail spatial characters of a leaky buffer that explain the mechanism through which these properties are produced.

6. REFERENCES

[1] 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.

[2] 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`.

[3] 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`.

[4] Steven H. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Trans. on Networking, to appear*, October 2003. `http://netlab.caltech.edu`.

[5] 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`.

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

[7] 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`.

[8] Rong Pan, Chandra Nair, Brian Yang, and Balaji Prabhakar. Packet dropping mechanisms: some examples and analysis. In *Proc. of 38th Annual Allerton Conference on Communication, Control, and Computing*, October 2001.

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

[10] 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.

[11] Ao Tang, Jiantao Wang, and Steven H. Low. Understanding CHOKe. In *Proc. of IEEE Infocom*, April 2003. `http://netlab.caltech.edu`.