Fair and efficient router congestion control

Xiaojie Gao* Kamal Jain† Leonard J. Schulman‡

Abstract

Congestion is a natural phenomenon in any network queuing system, and is unavoidable if the queuing system is operated near capacity. In this paper we study how to set the rules of a queuing system so that all the users have a self-interest in controlling congestion when it happens.

Routers in the internet respond to local congestion by dropping packets. But if packets are dropped indiscriminately, the effect can be to encourage senders to actually increase their transmission rates, worsening the congestion and destabilizing the system. Alternatively, and only slightly more preferably, the effect can be to arbitrarily let a few insistent senders take over most of the router capacity.

We approach this problem from first principles: a router packet-dropping protocol is a mechanism that sets up a game between the senders, who are in turn competing for link capacity. Our task is to design this mechanism so that the game equilibrium is desirable: high total rate is achieved and is shared widely among all senders. In addition, equilibrium should be reestablished quickly in response to changes in transmission rates. Our solution is based upon auction theory: in principle, although not always in practice, we drop packets of the highest-rate sender, in case of congestion. We will prove the game-theoretic merits of our method. We'll also describe a variant of the method with some further advantages that will be supported by network simulations.

1 Introduction

In a packet-based communication network such as the internet, packets go through routers on their way from source to destination. A router must examine each packet header and perform certain operations including, most significantly, deciding along which of the physical links the packet should be sent. In many (though not all) cases, this processing of the header is the limiting factor determining the capacity (in packets per second) of the router. In order to accommodate traffic bursts, the router maintains a queue. However, when traffic arrives over a sustained period at a rate above the router capacity, the length of the queue will approach the available buffer size; eventually, packet loss is unavoidable.

In the TCP protocol, packet losses (when communicated back to the source by the absence of acknowledgments) cause the source to reduce its transmission rate. This "backoff" mechanism has shown itself to be remarkably successful at maintaining a functional internet in spite of congestion. However, not all TCP installations implement this backoff policy. Moreover, not all traffic in the internet is generated by TCP; most significantly, UDP, which increasingly carries voice and video signals, is unresponsive, which is to say, it does not have a backoff mechanism. There is nothing inherent in the design of the internet to impose a charge or penalty on unresponsive traffic in order to discourage it from crowding out responsive traffic. This raises the specter of internet performance degrading due to high-volume unresponsive traffic — in turn causing more users to implement unresponsive transmission policies. The quality of service properties of such a network are not likely to be desirable, and include the possibility of *congestion collapse* in the internet.

Naturally this problem has drawn significant attention in the networking literature. For a better introduction than we could possibly provide here, see [4] as well as [17] and [14]. In the sequel we will not dwell further on the motivation, but adopt the problem from the existing literature, and focus on the technical aspects of prior solutions, their advantages and limitations — finally pointing out a significant limitation to all existing solutions, and our approach to addressing it.

Comment: Router design is an active field; modern high-speed routers are complex and contain several processing units and buffers. We follow prior literature in using the simplified one-processor one-queue model as a representative for whichever component of the router happens, in a given circumstance, to be the bottleneck.

The basic problem

The engineering challenge is to design a congestion control mechanism — to be implemented at routers, since we cannot control the sources — to achieve the following simultaneous objectives.

1. The rate (packets transmitted per second) of a router should at all times be close to the lesser of the router capacity and the received traffic.

2. The achieved rates of the various sources should be fair: high-volume sources should not be able to

*Computer Science Department, California Institute of Technology. Email: xiaojie@caltech.edu

†Microsoft. Email: kamalj@microsoft.com

‡Computer Science Department, California Institute of Technology. Supported by National Science Foundation grant no. 0049092 and by the Charles Lee Powell Foundation. Email: schulman@caltech.edu

Copyright © 2004 by the Association for Computing Machinery, Inc. and the Society for industrial and Applied Mathematics. All Rights reserved. Printed in The United States of America. No part of this book may be reproduced, stored, or transmitted in any manner without the written permission of the publisher. For information, write to the Association for Computing Machinery, 1515 Broadway, New York, NY 10036 and the Society for Industrial and Applied Mathematics, 3600 University City Science Center, Philadelphia, PA 19104-2688

crowd out low-volume sources.

This challenge has been taken up in several papers. "Fair queuing" mechanisms create separate queues for the packets from each source, and generally forward packets from different sources in round-robin fashion, selectively dropping packets from high-volume sources in order to fairly allocate rate among sources [3, 15, 7, 16, 2]. An attempt is usually made to allocate to each source its max-min-fairness rate, defined as follows:

DEFINITION 1.1. If the desired rate of each source i is r_i, and the router capacity is C, then the max-min-fairness rates are $R_i = \min\{r_i, \alpha\}$ where $\alpha = \alpha(C, \{r_i\})$ is the supremum of the values for which $\sum R_i < C$. (This includes the possibility $\alpha = \infty$ if $\sum r_i < C$.)

The fair queuing (FQ) proposals, however, have been criticized as computationally too intensive. While the per-packet computations involved are straightforward, it must be kept in mind that processing by the CPU for the purpose of congestion control, comes at the expense of time spent processing packet headers, and therefore, at the expense of router rate. (It might be suggested to use an extra CPU for the bookkeeping, but then the performance of the system should be compared with that of two routers.) A proposal has been made to economize on the computations by hashing and using fewer queues [12], but since many queues (perhaps thousands) will still be required in this method, the essential difficulty, that objective (2) is achieved at the expense of objective (1), persists.[1]

Other proposals have attempted to come at the problem by less complex modifications of the basic "FIFO with drop tail" queue that is at present most commonly used in the internet. (A single FIFO queue maintained for all packets, with packets at the end being dropped when buffer size is exceeded.) In "ERD" (early random drop) [8] and "RED" (random early detection) [5] packets are dropped at random when the queue lengthens; this is computationally very easy to implement but does not achieve objective (2). In [11] a modification of these methods, FRED (flow random early drop), was proposed in which packets are dropped with probabilities that depend on the flow volumes of their sources, in order to enforce roughly fair throughputs. While the absence of separate queues for each flow, in this method, represents a computational improvement over the FQ methods, the need to maintain separate bookkeeping for each flow subjected this proposal to similar criticism of excessive computational overhead.

To address this issue, it was proposed in [17] that "core" routers of the network adopt a protocol that does not maintain "state" (such as a record of the volume) for each flow. However, such per-flow states still need to be maintained at "edge" routers of the network, which then need to communicate rate estimates for the various flows to the core routers. In this method the core routers are not slowed down by flow-specific computations. Like the FQ methods, this proposal aims to achieve fair usage of router capacity by explicitly allocating to each flow its max-min-fairness rate. Potential drawbacks of this method are the assumption that there are edge routers with excess computational capacity (which begs the question of whether that capacity, or the resources to create that capacity, would not be better employed elsewhere), as well as the potential vulnerability or instability of a method that depends on message-passing between routers, in comparison with methods that are implemented independently at each router.

Allocations vs. penalties

The above approaches can be thought of as "allocation" methods in the sense that the router does its best to allocate to each source its max-min-fairness share of the router capacity; packets sent above that share are simply dropped. There has also been some work on what we'll call "penalty" methods, in which the router tries to discourage aggressive behaviour of sources, by actively penalizing sources that transmit more than their share. The advantage of a penalty system is that it motivates socially responsible behaviour, and thereby may reduce labor on the part of the router. (In a pure allocation scheme, there is no penalty for unresponsive behavior, so long as the traffic is robust to packet losses. Any traffic can be encoded so that this is the case. (The priority-encoded transmission methods of [1] show, more generally even than needed here, how to encode data so that no matter which packets are received, the highest-priority bits of the data can be recovered at a rate that is almost proportional to the number of packets received.)

Some suggested penalty methods, and the general advantages of employing penalties, were discussed in [4]. In the "Stable RED" (SRED) proposal [13], a fairly simple method was suggested to identify high-volume flows, and penalize them by preferentially dropping their packets; this idea was further simplified, both from a conceptual and computational point of view, in the CHOKe proposal [14]. The basic idea is that when the queue is long, each incoming packet is compared against another randomly selected packet; if they are from the same source, both are dropped. This has the merit of pref-

[1] In some cases the bottleneck on router capacity is not header processing time, but the I/O rate limit for sending the packet data. In such cases, fair queuing or other computationally intensive methods may be practical.

erentially penalizing high-volume flows. Moreover (like the other RED variations, as well as the FQ methods), it can be implemented at any router, independently of other routers. Although CHOKe's performance is not well understood analytically, extensive simulations have provided support for its favorable performance with regard to both objectives (1) and (2) above, *provided there is just one UDP (unresponsive) flow and all other flows are TCP compliant* [18].

On the other hand when several flows are unresponsive, it's easy to see that CHOKe will fail to prevent those from crowding out the responsive flows. Roughly speaking, if a flow occupies fraction p_i of the incoming traffic to the router, then fraction p_i of its packets will be dropped by the router; this prevents a single unresponsive flow from trying to dominate traffic into the router, but if there are even two unresponsive flows, they have no incentive to leave any capacity to the responsive flows. If some bound can be assumed on the number of unresponsive flows, then this problem can be compensated for by increasing the complexity of CHOKe: for instance, by sampling a set of more than just two packets, and deleting any packets that occur multiply in the set. However, since the size of this set needs to grow at least linearly with the bound on the number of unresponsive flows, the complexity of this solution grows sharply with that bound, and the solution loses its principal merit, the computational efficiency that yields objective (1).

Our contribution

There is no reason to suppose that, in practice, the number of flows aggressively (unresponsively) maximizing their throughput will be bounded by one, or by any other small number. This limitation of CHOKe is the stimulus for our contribution. Our approach is rooted in game theory and, in particular, in what is known as mechanism design. The perspective is that as the designer of the router protocol, we are in charge of a game among the sources, each of which is trying to achieve throughput as close as possible to its desired transmission rate. It is well known that, under certain technical conditions, such multiplayer games have Nash equilibria in which the strategies chosen by each of the players, are best possible conditional on the strategies of the other players. It is our task to set up the game so that its Nash equilibria satisfy our design objectives (1) and (2).

We begin now by specifying the technical requirements we demand of our solution. There are two types of requirements: (A) Computational requirements, (B) Game-theoretic requirements.

(A) Computational requirements:

A1. The per-packet time complexity of implementing the router protocol should be constant. (A small constant, comparable with CHOKe.)

A2. The space complexity of implementing the router protocol should be within a constant factor of simply maintaining a single packet queue.

A3. The protocol should be deployable at a single router, with no dependence on whether it has been deployed at any other routers.

(B) Some explanation is needed before presenting the game-theoretic requirements. We will not try to apply the theory of Nash equilibria to the most general situation in which there are infinitely many sources, each sending messages at times entirely of their choosing, in full knowledge of the randomized strategies of every other source. In view of the very little information actually available in practice to each source, and the overall asynchrony in a large network, this is a needlessly general setting. Instead, we will start with the case in which every source is Poisson. After establishing the basic game-theoretic conclusions in this framework, we will go on to consider what one source can gain by deviating from this framework, if all the *other* sources remain Poisson. (This is not a severe restriction because even if the other sources send packets at deterministic times, network delays on the way to the router introduce noise into the arrival times.) While the Poisson model is not good for short bursts of traffic, it is a reasonable model, much used in the networking literature (in spite of some limitations) for aggregate and extended-duration traffic.

We stress that when considering a source that is trying to "trick" our router, we will not constrain that source to generate Poisson traffic; the source will be allowed to generate traffic in an arbitrary pattern.

Notation: Let r_i be the desired transmission rate of source i. Let C be the capacity of the router. (Specifically, C is the rate the router can achieve while administering a single queue and spending constant time per packet on congestion control. Equivalently, the rate achievable by CHOKe.) Let R_i be the max-min-fairness rate of the source i (as defined earlier), given $\{r_i\}$ and C. Let s_i be the actual Poisson rate chosen by source i. Let a_i be the throughput of source i.

Our game-theoretic requirements are:

B1. Assume an idealized situation in which the router, and all the sources, know the source rates $\{s_i\}$; and in which the queue buffer is unbounded. This idealized game should have a unique Nash equilibrium which is the max-min-fairness rates R_i as determined by the inputs C and $\{r_i\}$.

As a corollary, when all sources are acting in their own best interest, the rate of the router equals C.

B2. In the actual game (which is administered by a router that can only use its history to govern its actions), there is a small $\varepsilon > 0$ such that any source sending at rate $s_i \leq (1-\varepsilon)\alpha$, will achieve throughput $a_i \geq s_i(1-\varepsilon)$.

As a corollary, when all sources are acting in their own best interest, the rate of the router is at least $C(1-2\varepsilon)$.

By establishing (B2), we will have accomplished the capacity and fairness objectives (1,2) specified earlier. This will be done in section 4.1.

The next step will be, as indicated earlier, to remedy (to a degree) our insistence on considering only Poisson sources. We will show:

B3. The long-term throughput of a source which is allowed to send packets at *arbitrary* times (while all other sources are still restricted to being Poisson) is no more than $1+\varepsilon$ times that of the best Poisson strategy.

Finally, we'll attend to the performance of a TCP source in our system. The reason for this is not game-theoretic; naturally, TCP is not likely to perform quite so well as a strategy optimized to play our game. Rather, the reason to consider TCP is that it is precisely the sort of responsive protocol which a mechanism such as ours is supposed to reward, and that it presently serves (according to [18]) at least 90% of internet traffic. Therefore it is important to show that TCP achieves good throughput in our system. In section 4.3 we'll show:

B4. Under certain assumptions on the timing of acknowledgments, the throughput of a TCP source with unbounded desired rate, playing against Poisson sources with desired rates $r_2, r_3, ...$, is within a constant factor of the max-min-fairness value $\alpha(C, \{\infty, r_2, r_3, ...\})$.

The reason that TCP interacts so well with our protocol is that it backs off very quickly from congestion, and therefore, will quickly stop being "punished" by our protocol; and that it subsequently "creeps" up toward the max-min-fairness threshold α (before again having to back off).

2 The protocol

From the network packet queuing theory perspective our protocol is similar to CHOKe. In case of congestion, CHOKe penalizes all the sources in proportion to their sending rate. We instead penalize only the highest rate senders. So all the senders compete to not be the highest rate flow; this eliminates the congestion.

One can also consider CHOKe and our protocol as unusual sorts of single-item auctions in which the players wish to bid as high as possible without actually winning the item. From this perspective the winner, in the case of CHOKe, is picked randomly with probabilities proportional to the bids; whereas in our protocol the winner is the highest bidder. Since nobody wants to "win" the penalty, the senders, in our protocol, have an incentive to lower their bids until the total is low enough that the auction is cancelled.

We'll actually describe two slightly different versions of the protocol. In either case, the protocol will maintain several items of data:

1. Q, the total number of packets presently in the queue.
2. A hash table containing, for each flow i having packets in the queue, a record of m_i, the total number of packets presently in the queue from source i.
3. *MAX*, a pointer to the record of the source having the highest number of packets in the queue.

In addition, there are several adjustable parameters controlling the protocol behavior: F, the size of the queue buffer; "high" H and "low" L markers satisfying $0 < L < H < F$.

The protocol is defined by the actions it takes when packets arrive at the router, and when they depart the head of the queue. We describe these separately. We begin with Protocol I, to which we'll address the theorems of this paper. Protocol II, given at the end of this section, is very similar, but is better at coping with multiple UDP sources, as will be illustrated by simulation in section 6.

Protocol I

Packet arrivals

Each time a packet arrives, the following actions are performed:

1. The packet source i is identified.
2. (a) If $Q > H$, stamp the packet *DROP*;
   (b) otherwise if $H \geq Q > L$ and $i = MAX$, stamp the packet *DROP*;
   (c) otherwise, stamp the packet *SEND*.
3. The packet is appended to the tail of the queue, together with the stamp. (If the stamp is *DROP*, the packet data can be deleted, but the header is retained so long as the packet is on the queue.)
4. Q and m_i are incremented by 1. (If m_i was 0, a new record is created.)

5. If $m_i > m_{MAX}$, then the MAX pointer is reassigned the value i.

Packet departures

Each time a packet is pulled off the head of the queue, the following actions are performed:

1. The packet source i is identified.
2. Q and m_i are decremented by 1. (If $m_i = 0$ the record is eliminated from the hash table.)
3. If the packet is stamped *SEND*, it is routed to its destination; otherwise, it is dropped.
4. If $i = MAX$ but m_i is no longer maximal, MAX is reassigned to the maximal sender.

Comment 1: the usual procedure when facing buffer overflow is to "droptail", i.e., to continue to serve packets already on the queue but not to accept new packets into the queue. Here, we make drop decisions at the tail of the buffer, but we don't really drop the packets there. Instead, we append a packet to the tail of the buffer together with the stamp of *SEND* or *DROP* and send or drop it at the head of the queue according to its stamp. It may appear peculiar, at first sight, that when we decide to drop packets we put them on the queue anyway, and only really get rid of them when they reach the head of the queue. The reason is that our queue serves a dual purpose. One is the ordinary purpose: a buffering facility to time-average load and thereby approach router capacity. The other purpose is as a measuring device for recent traffic volumes from the sources. In our method the contents of the queue represent, in all circumstances, the complete history of packets received at the queue over some recent interval of time.

Our handling of drops enables us to use the $\{m_i\}$ to estimate the source rates instead of having to compute exponential averages, as was done in some of the recent literature in this area. (The averaging is not complicated but requires reference to the system clock plus some arithmetic; in view of the computational demands on the router, the gain may be meaningful.)

Comment 2: Since we don't "droptail", we need to ensure that we don't create buffer overflow. There are three time parameters associated with this queue: T_1 = the time to route a *SEND* packet at the head of the queue, T_2 = the time to append a packet to the tail of the queue, T_3 = the time to move the head of the queue past a *DROP* packet. For a queuing system to make sense, these times should satisfy the inequalities $T_1 > T_2 > T_3$. We can prevent buffer overflow in our protocol simply by choosing F and H so that $F/H \geq T_1/T_2$. (This is not hard to show.)

Protocol II

Protocol II differs from Protocol I only in line 2(b) of **Packet arrivals**, which we replace by:

2(b)' If $H \geq Q > L$ and $m_i \geq \frac{H-Q}{H-L} m_{MAX}$, stamp the packet *DROP*; otherwise, stamp it *SEND*.

Protocol II has no advantage over Protocol I with regard to Nash equilibria. However, its sliding scale for drops has a substantial advantage over both CHOKe and Protocol I in the effectiveness with which multiple unresponsive flows are handled. This will be demonstrated by simulation in section 6.

3 Satisfaction of the computational requirements (A)

The most straightforward way to handle the protocol computations is to maintain the active sources (those with $m_i > 0$, i.e., those having packets in the queue) in a priority queue, keyed by m_i. This does not entirely resolve the computational requirements, though, for two reasons: (a) updates to a priority queue with n items take time $O(\log n)$, rather than a constant. (b) A hashing mechanism is still required in order to find, given a source label i, the pointer into the priority queue.

Item (a) is easily addressed. Since we change m_i by only ± 1 in any step, the following data structure can substitute for a general-purpose priority queue. Maintain, in a doubly-linked list, a node N_k for each k such that there exists i for which $m_i = k$. The linked list is maintained in order of increasing k. At N_k maintain also a counter $c(k)$ of the number of distinct i for which $m_i = k$. Finally, from N_k maintain also $c(k)$ two-way pointers, each linking to a node at which one of those labels i is stored. This data structure can easily be updated in constant time in response to increments or decrements in any of the m_i.

Item (b) is slightly more difficult since it asks, essentially, for a dynamic hash table with $O(1)$ access time and linear storage space. (The elegant method of Fredman, Komlós and Szemerédi (FKS) [6] is not dynamic.) We know of no perfect way to handle this, but two are at least satisfactory.

One solution is to modify FKS, allow polylogarithmic space overhead, and achieve constant amortized access time by occasionally recomputing the FKS hash function.

An alternative solution is simply to store the pointers in a balanced binary search tree, keyed by the source labels $\{i\}$. This method uses linear space but $O(\log n)$ access time. A simple device fixes the access time problem: instead of updating m_i and Q with every packet, perform these updates only every $(\log F)$'th packet. Our game-theoretic guarantees will still hold with a slight

loss in quality due to the slightly less accurate estimation of flow rates, and with slightly slower reactions to changes in flow rates. In practice we anticipate that these effects will be negligible, and therefore that this is preferable to the modified-FKS solution.

In some operating environments there might be a moderate, known bound on the number of sources whose rates are close to maximal. In such cases it may be possible to take advantage of an attractive method [10] which keeps track of the k most-frequent sources in a stream, using only memory $O(k)$. (However the technique tracks the statistics of the entire, rather than only the recent, history; so some finite-horizon version would have to be adopted.)

4 Satisfaction of the game-theoretic requirements (B)

We assume there are B Poisson sources and their Poisson rates are s_i with $s_1 \geq s_2 \geq s_3 \geq \cdots \geq s_B$. Let $\widetilde{B} = \min_j\{\sum_{i=1}^{j} s_i \geq \frac{1}{2}\sum_{i=1}^{B} s_i\}$. Observe that $\widetilde{B}$ is an undercount of the number of sources: it omits sources generating very sparse traffic, and counts only the number of sources contributing a substantial fraction of the total traffic.

4.1 Equilibrium guarantees: properties B1, B2

We first consider a "toy" version of our protocol in which all the sources, and the router, know the true transmission rates s_i of all sources. (As indicated earlier, we'll assume the sources are constant-rate Poisson.) Moreover, the buffer for the queue is unbounded. If $\sum s_i > C$, the router simply drops all the packets of the highest-rate source; if several tie for the highest rate, it rotates randomly between them.

THEOREM 4.1. *If the sources have desired rates $\{r_i\}$ for which $\sum r_i > C$, and can transmit at any Poisson rates $0 \leq s_i \leq r_i$, then their only Nash equilibrium is to transmit at rates $s_i = R_i$, for R_i the max-min-fairness rates.*

Proof. This is immediate (recalling that we treat only the case of finitely many sources). If $\sum s_i < C$ then naturally some source can improve its rate. If $\sum s_i \geq C$ and any of the source rates exceeds α, then suppose k of them tie for the highest source rate s, i.e., each of those k sources has the largest number of packets in the queue. The throughputs of those k will be $s(1-1/k)$, whereas they could have done better by transmitting at a rate slightly less than s. Therefore no source rate can exceed α. □

This establishes game-theoretic property B1. The rest of this section is devoted to property B2: showing that the above argument survives the constraint of having to make do with a finite queue buffer. We first need a Chernoff bound for the probability that a Poisson process deviates far from its mean.

LEMMA 4.1. *X is a Poisson process with $E[X] = \lambda$:*

$$\begin{aligned} Pr\{X \geq (1+\delta)\lambda\} &< e^{-\lambda\delta^2/4}, \text{ for } \delta \in (0, 2e-1); \\ Pr\{X \geq (1+\delta)\lambda\} &< 2^{-\lambda\delta}, \quad \text{for } \delta \in [2e-1, +\infty); \\ Pr\{X \leq (1-\delta)\lambda\} &< e^{-\lambda\delta^2/2}, \text{ for } \delta \in (0, 1]. \end{aligned}$$

Proof. Details omitted. □

LEMMA 4.2. *X and Y are two Poisson processes with $E[X] = \alpha$, $E[Y] = \beta$, and $\alpha < \beta$:*

$$Pr\{X \geq Y\} < e^{-\frac{\alpha(\beta-\alpha)^2}{4(\beta+\alpha)^2}} + e^{-\frac{\beta(\beta-\alpha)^2}{2(\beta+\alpha)^2}}.$$

Proof. Details omitted. □

If we assume the buffer is large enough so that $L \in \Omega(\widetilde{B}\frac{1}{\varepsilon^2}\ln\frac{1}{\varepsilon})$, we have the following theorem.

THEOREM 4.2. *If the protocol of section 2 is administering traffic from sources with desired rates $\{r_i\}$ for which $\sum r_i > C$, and which can transmit at any Poisson rates $0 \leq s_i \leq r_i$, there is a small $\varepsilon > 0$ such that any source sending at rate $s_i \leq (1-\varepsilon)\alpha$, will achieve throughput $a_i \geq s_i(1-\varepsilon)$. (Here $\alpha = \alpha(C, \{r_j\})$ and $L \in \Omega(\widetilde{B}\frac{1}{\varepsilon^2}\ln\frac{1}{\varepsilon})$.)*

Proof. The hypotheses guarantee a ratio of at most $1-\varepsilon$ between s_i and $\alpha(C, \{s_j\})$. Packets from source i will be dropped only when the queue grows to size L, and m_i is higher than that of all other sources.

Case 1: $\sum_j s_j \geq \sum_j \min\{(1-\varepsilon/2)\alpha, r_j\}$. The idea for this case is that with high probability the packet dropping ends quickly, since the protocol will soon identify a higher-rate source.

In this case s_i is not the largest, and there must be a fluctuation in the transmission rates that causes m_i to be the largest. Assume source k is sending at the largest rate s_k. It's clear that $s_k \geq (1-\varepsilon/2)\alpha$ and $s_i \leq (1-\frac{\varepsilon/2}{1-\varepsilon/2})s_k$. Let $\delta = (s_k - s_i)/(s_i + s_k) \geq \varepsilon/(4-3\varepsilon)$; $\delta_0 = (\sqrt{5}-1)/2$ when $\delta \geq (\sqrt{5}-1)/2$, otherwise $\delta_0 = \delta$. Suppose the time interval between the arrival time of the packet at the head of the queue and that of the packet at the rear is t, so $Pr\{m_i = m_{MAX}\} \leq Pr\{m_i\frac{(1-\delta_0)(1+\delta)}{(1+\delta_0)(1-\delta)} \geq m_k\} < 2e^{-s_k t(1-\delta_0)\delta_0^2/(4(1+\delta_0))} < \varepsilon$. (Details omitted.) The probability that a packet from source i is dropped is less than ε. So source i will achieve throughput $a_i \geq s_i(1-\varepsilon)$.

Case 2: $\sum_j s_j < \sum_j \min\{(1-\varepsilon/2)\alpha, r_j\}$. In this case the cause of the losses is a fluctuation causing the

queue length to increase to L. The fraction of packets lost due to this reason is proportional to the probability of such a fluctuation. The argument proceeds by showing that if L is large enough, this probability is very small. (Details omitted.) □

4.2 Sources cannot increase throughput by variable-rate transmission: property B3 Consider the case that the transmission rates of all sources except one (denote it source S) are fixed. We have the following theorem:

THEOREM 4.3. *For a source which is allowed to send packets at arbitrary times (while all other sources are restricted to being Poisson), the long-term throughput is no more than $1+\varepsilon$ times that of the best Poisson strategy.*

Proof. Assume source S is allowed to send packets at arbitrary times while other sources $1, 2, \cdots, B$ are restricted to being Poisson process with rates s_i ($s_1 \geq s_2 \geq \cdots \geq s_B$).

Consider the packets from source S arriving in $[T, T']$. We ignore the last a few packets which are dropped as they have no contribution to the throughput. Look at the last packet p_1 from source S which is not dropped. Let its arrival time be t_1 and at time t_1, the arrival time of the packet at the head of the queue be t'_1. Since packet p_1 is stamped *SEND*, in the time interval $[t'_1, t_1]$, and so in $[t'_1, T']$, the number of routed packets from S is no more than m_{MAX} at time t_1; otherwise, the stamp will be *DROP*. By the same argument, we can split the interval $[T, T']$ into $n+1$ sub-intervals: $[T, t'_n]$,$[t'_n, t'_{n-1}]$, $[t'_{n-1}, t'_{n-2}], \cdots, [t'_2, t'_1], [t'_1, T']$. Let $t'_0 = T'$ and $m_{MAX} = MAX_i$ at time $t'_i (i \in \{0, 1, \cdots, n-1\})$. The number of routed packets from S in $[t'_n, T']$ is no more than $MAX_0(t'_0 - t'_1) + MAX_1(t'_1 - t'_2) + \cdots + MAX_{n-1}(t'_{n-1} - t'_n)$. Given that $[T, T']$ is long enough, the throughput of source S in $[T, T']$ is approximate to that of source S in $[t'_n, T']$.

Since in any $[t'_{i+1}, t'_i]$, $E[MAX_i] \leq s_1(t'_i - t'_{i+1})(1+\varepsilon)$ (Details omitted), the number of routed packets from S, in the interval $[T, T']$, is no more than $1+\varepsilon$ times that of arrival packets from source 1. The best throughput for source S is no more than $1+\varepsilon$ times that of the Poisson strategy with rate s_1. □

4.3 Performance of TCP: property B4 We give here a brief overview of TCP from a theoretical perspective. TCP is a transmission protocol whose main idea is additive increase and multiplicative decrease in rate in response to absence or presence of congestion. TCP maintains a rotating window of a fixed size, say N, at the sender side. The rotating window is basically a set of N buffers named in a circular manner. When a packet arrives (from some application) at a sender, the packet is parked into one of the available buffers and also sent over the network. If no buffer is available then the packet generation rate is higher than the serving rate. In this case the rate is halved. The parked packets are removed once the acknowledgement of their successful receipt is received. If all the buffers are emptied then the packet generation rate is smaller than the serving rate. In this case the rate is increased by a constant, say 1. If the generation rate is exactly the same as the serving rate then the buffer will reach the empty state or full state occasionally. Since in our protocol the ideal serving rate, which is given by the "max-min-fairness" threshold α, is not precisely known, we assume that the generation rate is not equal to the serving rate. (Also, equality is actually a favorable case; we write the following theorem from the worst-case point of view.) Let T^I_{TCP} be the time to increase the generation rate from 0 to α, or from α to 2α, if all the packets are getting through; let T^D_{TCP} be the time to decrease the generation rate to 0, starting from rate at most 2α at the moment that all packets begin to be dropped; and let T^W_{TCP} be the waiting time until the generation rate starts increasing, once the router begins allowing packets through. These parameters are illustrated in Figure 1.

THEOREM 4.4. *If $T^D_{TCP}, T^W_{TCP}, T_1F \in O(T^I_{TCP})$, then the throughput of an adaptive flow using a TCP-like adjustment method of increase-by-one and decrease-by-half in our system is at least optimal/D for some constant D.*

Proof. We'll suppose in order to simplify details that $T^D_{TCP}, T^W_{TCP}, T_1F \leq T^I_{TCP}$.

Examine the router starting at a moment at which the TCP generation rate is smaller than the serving rate, which is the "max-min-fairness" threshold α. So at this time, the rate is increased by 1. When the number of packets from the flow becomes maximal among all flows (at which time the rate is between α and 2α, since $T_1F \leq T^I_{TCP}$), the arriving packets will start to be dropped. Then within time T^D_{TCP}, the sender will have reduced its transmission rate. After an additional time at most T_1F, the queue will have cleared, its statistics will reflect the low transmission rate and the router will stop dropping packets from this source. Finally after an additional time at most T^W_{TCP}, the generation rate will again begin increasing. The worst-case throughput for the flow is given by a history as in Figure 1, in which the curve is the generation rate as a function of time and the area in shadow illustrates the throughput of the flow (We have pessimistically supposed even that TCP backs

off all the way to rate 0). Given the simplified timing assumptions, the worst-case throughput achieves $D \leq 8$. □

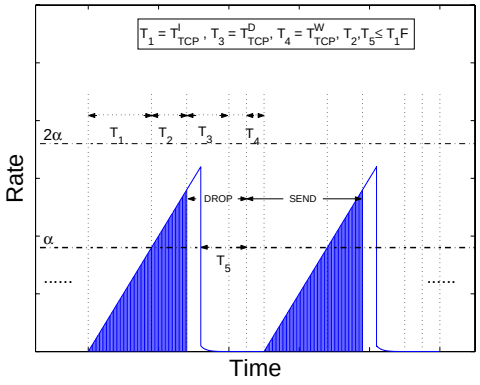

Figure 1: The generation rate and throughput of the adaptive flow using a TCP- like adjustment method of increase-by-one and decrease-by-half, as a function of time.

As a final remark of this section, the use of a TCP-like adaptive mechanism is well motivated in our protocol. The protocol punishes a violating flow heavily, so it is a priority for such a flow to come as quickly as possible to below the "max-min-fairness" rate α. Multiplicative decrease is a good way to do so. Once the rate is below α, the next priority is to optimize the flow by gradually increasing it, without overshooting.

5 Network equilibrium

There are generally many routers in a network. We wish to understand what the Nash equilibria are in the case of a general network, with several flows traveling across specified routes, and with protocol I implemented at each of the routers of the network.

We will show that — at least under stable traffic conditions and given accurate estimation of source rates (just as for property B1) — the favorable properties of protocol I extend to this general network case. Jaffe has shown [9] that there for any set of routes in a capacity-limited network there is a unique *max-min-fairness* flow, which shares network capacity as evenly as possible among the flows. We will show that when the routers use protocol I, the sources have a unique Nash equilibrium, which is none other than the max-min-fairness flow.

We begin by pointing out that in any equilibrium, it is to the advantage of every source to be transmitting at no more than its throughput. The principal reason is that if packets are being dropped, the message must be encoded to be reconstructible from the random set of packets which get through, and that the message rate must therefore be somewhat lower than the throughput rate. Therefore by reducing transmission rate until no packets are being dropped, a source can still get just as many packets through, and increase its message rate. (This argument doesn't apply to low-volume sources whose throughputs are not limited by the network; but even for such sources, there are small additional costs, e.g., in CPU time, associated with generating extraneous packets.)

This reduces our task to showing:

LEMMA 5.1. *If each flow is sending at its throughput, then there is a unique Nash equilibrium, the "max-min-fairness" allocation.*

Proof. It is well known that the "max-min-fairness" allocation is unique, so we have noly to show that any Nash equilibrium is max-min-fair.

For each router, the summation of the "max-min-fairness" rates for all flows going through the router is at most its capacity. Assume $\overrightarrow{x}$ is a Nash equilibrium and $\overrightarrow{y}$ is any other allocation. If there exists an $s \in \{1, \ldots, N\}$(where N is the number of flows) such that $y_s > x_s$, then $x_s < r_s$ (where r_s is the desired rate of flow s). Hence, there exists a router A carrying flow s and such that the bandwidth of any other flow on A is at most x_s. (Otherwise, for any router, x_s is not the highest rate it carries, so x_s can be increased and $\overrightarrow{x}$ is not a Nash equilibrium.) So there exists a $t \in \{1, \ldots, N\}$, $t \neq s$, such that $y_t < x_t \leq x_s$. Therefore, $\overrightarrow{x}$ is the "max-min-fairness" allocation. □

Let us formalize the preceding discussion by saying that the utility function of each transmitter is its throughput, less some small (even infinitesimal) multiplier times the number of its packets that are dropped. Under this assumption we have:

THEOREM 5.1. *For any collection of network flows there is a unique Nash equilibrium, equal to the "max-min-fairness" allocation; in this equilibrium there are no packet drops.*

6 Simulations

We evaluate the performance of our protocol by simulations under various network configurations and parameters. Overall, Protocol I and II achieve reasonably fair bandwidth allocations. Protocol II does particularly well at handling several unresponsive sources (this case is not covered by studying Nash equilibria, since each source would do better by backing off a little). Some additional simulations will be posted at www.cs.caltech.edu/∼xiaojie/Research/Simulation/ results.html.

6.1 Simulation 1: testing the equilibrium property To verify the Nash equilibrium property of our protocol, we consider a single r Mbps congested link shared by 5 Poisson flows, indexed from 0 to 4. The sending rates of flows 0, 1, 2, and 3 are fixed: r Mbps, $2r/3$ Mbps, $r/2$ Mbps, and $2r/5$ Mbps. Flow 4 is allowed to send packet at arbitrary rates. We vary the sending rate of flow 4 to study our protocol over a $5000r$ sec interval. When the buffer size is set to be finite, $H = F/6$, and L is 0, the simulation results for Protocol I and II are summarized in Figure 2. When the sending rate reaches r, a large percentage of the packets are dropped. The greatest bandwidth, approximately $0.3r$ Mbps, is obtained at a sending rate less than r Mbps.

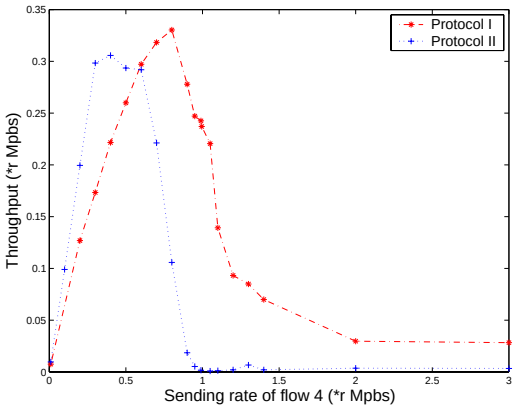

Figure 2: Average throughput of flow 4, as a function of the sending rate, under Protocol I and II.

In the following simulations we use ns-2 (a network simulator) to test more complicated behaviors.

6.2 Simulation 2: Performance on a single congested link

6.2.1 Comparison with CHOKe To compare the performances of Protocol I and CHOKe on a single congested link, we consider a 2 Mbps link shared by one UDP flow which is sending at 3 Mbps, indexed 1, and 32 TCP flows, indexed from 2 to 33, whose propagation delays are 3 ms. The average throughput of each flow over a 100 sec interval is given in Figure 3. Under Protocol I, the ill-behaved UDP flow is penalized heavily and the TCP flows get approximately their fair rates (as would happen without any penalties to unresponsive flows); but CHOKe doesn't provide as much bandwidth to each TCP source as to the UDP source.

6.2.2 Ten UDPs and ten TCPs flows on a single congested link To examine the impact of a set

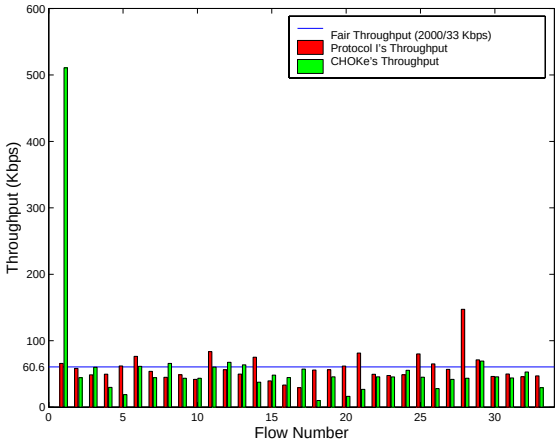

Figure 3: The average throughput of each flow over a 100 sec interval under Protocol I and CHOKe.

of ill-behaved UDP flows on a set of TCP flows, we consider a 1.5 Mbps link shared by ten UDP sources, each of which are sending at 0.5 Mbps, and ten TCP sources, which are using TCP Tahoe and whose propagation delays are set to 3 ms. (Simulations of TCP Reno, Newreno and Sack gave similar results.) We compare the performances of Protocol I, Protocol II, and CHOKe. The maximal and minimal throughputs of two kinds of flows over a 100 sec interval is given in Table 1. All of the protocols prevent the TCP sources from being entirely shut out of the channel (as would happen without any penalties to unresponsive flows); but Protocol I and CHOKe don't penalize the ill-behaved UDP flows enough as they still get more than their fair rates, and Protocol II stands out by providing just as much bandwidth (actually more) to the TCP sources than to the UDP sources.

Protocol	CHOKe	I	II
UDP MAX (Mbps)	0.15432	0.15576	0.00488
UDP MIN (Mbps)	0.14608	0.14376	0.00352
TCP MAX(Mbps)	0.00200	0.00224	0.14544
TCP MIN(Mbps)	0.00008	0.00056	0.13560

Table 1: The maximal and minimal throughputs of two kinds of flows over a 100 sec interval under Protocol I, Protocol II and CHOKe.

6.3 Simulation 3: Multiple Congested Links So far we have verified the Nash equilibrium and seen the performance of flows on a single congested link. We now analyze how the throughputs of flows are affected when they traverse more than one congested link. A simple

network configuration with three routers is constructed as shown in Figure 4. Each of the congested links has capacity 10 Mbps: the link between routers 1 and 2 (L12), and the following link between routers 2 and 3 (L23).

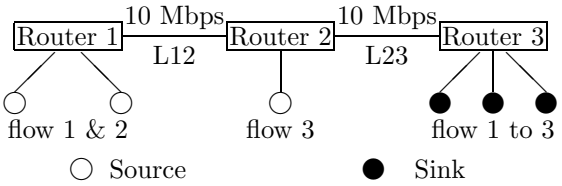

Figure 4: Topology for analyzing how the throughput of a flow is affected by more than one congested link.

There are three flows in the network: source 1 is a UDP source, which sends at 10 Mbps, and the other two are TCP sources. The propagation delays of TCPs are 3 ms. Using Protocol I, the average throughput of each flow over a 20 sec interval at various link in the network is listed in Table 2. Since there are two links, the throughput of TCP2 is dominated by link L23. The average bandwidth of TCP2 on Link L23 is 2.451 Mbps, it is reasonable that three quarters of the capacity of link L12 is occupied by UDP1. Both TCP flows get rates close to their fair rates on link L23; since there is only one non-adaptive flow (UDP flow 1), it also gets close to its fair rate on the link. The average throughputs of flows at various link in the network under Protocol II and CHOKe are also listed in Table 2. The performance of Protocol II is comparable to that of Protocol I and it outperforms CHOKe. Comparing these results with the single congested link case, we can see that non-adaptive flows, like UDP, suffer from more severe packet loss than adaptive flows.

	CHOKe		Protocol I		Protocol II	
(Mbps)	L12	L23	L12	L23	L12	L23
UDP1	9.579	8.943	7.339	4.125	7.643	4.077
TCP2	0.404	0.400	2.534	2.451	2.244	2.163
TCP3	-	0.615	-	3.196	-	3.530

Table 2: One UDP flow and two TCP flows. The average bandwidth of each flow is given under Protocol I, Protocol II, and CHOKe.

Acknowledgments: Thanks to Scott Shenker for helpful discussions and for access to simulation codes; to Steven Low and Ao Tang for helpful discussions; and to Jiantao Wang for access to simulation codes.

References

[1] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan. Priority encoded transmission. *IEEE Trans. Information Theory*, 42(6):1737–1744, 1996.

[2] J.C.R. Bennett and H. Zhang. WF^2Q: Worstcase fair weighted fair queueing. In *Proc. IEEE INFOCOM*, pages 120–128, 1996.

[3] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking Research and Experience*, pages 3–26, October 1990. Also in Proceedings of ACM SIGCOMM'89, pp 312.

[4] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Trans. on Networking*, 7(4), 1999.

[5] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397–413, 1993.

[6] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with O(1) worst case access time. *J. Assoc. Comput. Mach.*, 31(3):538–544, 1984.

[7] S. Golestani. A selfclocked fair queueing scheme for broadband applications. In *Proc. IEEE INFOCOM*, pages 636–646, 1994.

[8] E. Hashem. Analysis of random drop for gateway congestion control. *MIT LCS Technical Report 465*, 1989.

[9] J. M. Jaffe. Bottleneck flow control. *IEEE Trans. on Comm.*, COM-29(7):954–962.

[10] R. M. Karp, C. H. Papadimitriou, and S. Shenker. A simple algorithm for finding frequent elements in streams and bags. Manuscript.

[11] D. Lin and R. Morris. Dynamics of random early detection. In *Proc. ACM SIGCOMM*, pages 127–137, 1997.

[12] P. McKenny. Stochastic fairness queueing. In *Proc. IEEE INFOCOM*, pages 733–740, 1990.

[13] T. Ott, T. Lakshman, and L. Wong. SRED: Stabilized RED. In *Proc. INFOCOM*, pages 1346–1355, 1999.

[14] R. Pan, B. Prabhakar, and K. Psounis. CHOKe: a stateless active queue management scheme for approximating fair bandwidth allocation. In *Proc. IEEE INFOCOM*, 2000.

[15] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. In *Proc. IEEE INFOCOM*, 1992.

[16] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proc. ACM SIGCOMM*, pages 231–243, 1995.

[17] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: achieving approximately fair bandwidth allocations in high speed networks. In *Proc. ACM SIGCOMM*, pages 118–130, 1998.

[18] A. Tang, J. Wang, and S. H. Low. Understanding CHOKe. In *Proc. IEEE INFOCOM*, 2003.