Router Congestion Control

Thesis by

Xiaojie Gao

In Partial Fulfillment of the Requirements
for the Degree of
Master of Science

California Institute of Technology
Pasadena, California

2004
(Submitted June 3, 2004)

Acknowledgements

It is my great pleasure to thank my advisor Professor Leonard J. Schulman. This thesis would never have existed without his help, support, inspiration, and guidance. To him, I offer my most sincere gratitude.

I wish to thank my fellow members of the theory group for their valuable discussions and helpful suggestions for my work.

I am grateful to Wonjin Jang for the fruitful and enjoyable discussions of some of these matters with him in the course of my research.

I owe a lot to Chih-Kai Ko, who have helped me in writing on this work and providing constructive suggestions.

In addition, my thanks go to Kamal Jain for the intuition for this work and for stimulating discussions that have influenced much of the work in this thesis, to Scott Shenker for helpful discussions and for access to simulation codes used in this thesis, to Steven Low and Ao Tang for helpful discussions, and to Jiantao Wang for access to simulation codes.

Special thanks to my officemates, Helia Naeimi and Mortada Mehyar, for their encouragement.

I am particularly indebted to my whole family for their love, encouragement, and support, especially my parents who have always been there to offer guidance for me.

I offer my deep thanks to all of my friends, both in US and in China, who have helped me in many aspects of daily life and study.

This work was supported by the National Science Foundation under grant no. 0049092 (previously 9876172), and by the Charles Lee Powell Foundation.

Abstract

Congestion is a natural phenomenon in any network queuing system, and is unavoidable if the queuing system is operated at capacity. In this thesis, 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.

Queueing system is a crucial component in effective router congestion control since it determines the way packets from different sources interact with each other. If packets are dropped by the queueing system indiscriminately, in some cases, the effect can be to encourage senders to actually increase their transmission rates, worsening the congestion, and destabilizing the system.

We approach this problem from game theory. We look on each flow as a competing player in the game; each player is trying to get as much bandwidth as possible. Our task is to design a game at the router that will protect low-volume flows and punish high-volume ones. Because of the punishment, being high-volume will be counter productive, so flows will tend to use a responsive protocol as their transport-layer protocol. The key aspect of our solution is that by sending no packets from high-volume flows in case of congestion, it gives these flows an incentive to use a more responsive protocol.

In the thesis, we will describe several implementations of our solution, and show that we achieve the desired game-theoretic equilibrium while also maintaining bounded queue lengths and responding to changes in network flow conditions. Finally, we accompany the theoretical analysis with network simulations under a variety of conditions.

Contents

Chapter 1

Introduction

1.1 Motivation

In a typical packet-based communication network environment, 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.

Data flows[1] originate from sources using different transport layer protocols. These protocols can be categorized based on their response to network congestion.

TCP, the prevailing transport layer protocol, is an example of a responsive protocol. Most practical implementations of TCP ease network congestion by a "backoff" mechanism [19]: a source will reduce its rate when it infers, through packet loss, that the network is unable to sustain the current sending rate. This "backoff" mechanism has shown itself to be remarkably successful at maintaining a functional internet in spite of congestion. But the efficacy of this approach depends on an elementary as-

[1]A data flow is a stream of packets which traverse the same route from the source to the destination and require the same grade of service at each router in the path. Each packet is uniquely assigned to a flow according to pre-specified fields in the packet header [36].

sumption [6]: *All (or a great majority of) flows are responsive (running basically the same "backoff" congestion avoidance mechanisms).* Since "the Internet is no longer a small, closely knit user community" [9] and not all flows are responsive, it is no longer possible to just rely on this "backoff" mechanism to avoid a congestion collapse.

An unresponsive protocol, like UDP, has no such "backoff" mechanism. The rise in demand of streaming multimedia content, much of which travels on UDP, has caused a large increase in unresponsive flows in the Internet [12]. To make matters worse, the inherent design of the current Internet imposes no penalty or charge on unresponsive traffic to discourage it from crowding out the responsive traffic. In turn, this may cause more users to implement unresponsive transmission policies which eventually lead to a congestion collapse of the network. Therefore, it is necessary to prevent this with an efficient congestion control mechanism that is fair to both types of network flows.

It has been observed that router congestion control mechanisms that aim at allocating bandwidth in a fair manner could help preventing congestion collapse greatly [27]. In the current network architecture, the most common queueing algorithm is first-come-first-serve (FCFS) [32], there are no concrete incentives for individual sources to use responsive protocols, and there are, in some cases, "rewards" for unresponsive sources in that they might receive a larger fraction of the link bandwidth than they would otherwise. For the health of the network, we need to design a router congestion control mechanism that can provide reasonably fair and efficient congestion control and may indeed encourage sources to use responsive protocols at the senders.

Naturally this problem has drawn significant attention in the networking literature. For a better introduction than we could possibly provide here, see [9] (as well as [5], [33], and [29]). 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 approaches 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 whatever component of the router happens, in a given circumstance, to be the bottleneck.

1.2 The basic problem

The engineering challenge is to design a congestion control mechanism — to be implemented at routers, since we cannot control the sources and sources don't have information about other sources against whom they are competing for bandwidth of the network — to achieve the following simultaneous objectives:

1. **Efficiency** — 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. We say that a router is operating *efficiently* if it is operating approximately at the lesser of the router capacity and the received traffic.

2. **Fairness** — The achieved rates of the various sources should be fair: high-volume sources should not be able to crowd out low-volume sources.
   We follow common network literature in equating fairness with "*max-min-fairness*" criterion [4, 31, 17, 16]. To put it more precisely:

   > **Definition 1** *If the arrival rate of each flow i is s_i, and the router capacity is C, the* ***max-min-fairness*** *rate of flow i is $S_i = \min\{s_i, \alpha^*\}$, where $\alpha^* = \alpha^*(C, \{s_i\})$ is the supremum of the values for which $\sum_i S_i \leq C$. (When $\sum_i s_i < C$, $\alpha^* = \max_i s_i$.) α^* is called the* ***max-min-fairness*** *threshold.*

   Given the arrival rates of a set of flows, the "*max-min-fairness*" rate of each flow is unique [31]. Given a concave "utility function" $f(x)$, the "*max-min-fairness*" solution, in the single-router case, maximizes $\sum_i f(x_i)$, where i ranges over the sources.

This challenge has been taken up in several papers. In the following, we will first introduce a classification of usually used methods, and then describe a few main prior solutions to the problem.

1.3 Allocations vs. penalties

Generally, there are two, but not necessarily mutually exclusive, categories of methods to solve the above problem: *allocation* methods and *penalty* methods.

In *allocation* methods, 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.

The allocation methods give no incentive for drop-tolerant flows to use a responsive congestion control protocol. Drop-intolerant flows do have such an incentive, but any traffic can be encoded so that this is the case: the priority-encoded transmission methods of [2] show 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.

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 fair share.

The advantage of a penalty system is that it motivates socially responsible behaviour, and thereby may reduce overhead labor on the part of the router. Some suggested penalty methods, and the general advantages of employing penalties, were discussed in [9].

1.4 Prior solutions

In this section, we will describe a few main prior solutions to the problem. Especially, we will point out their advantages and limitations.

1.4.1 Fair Queueing

Demers, Keshav and Shenker proposed an isolation mechanism called ***Fair Queueing*** (FQ) in [7]. It uses Nagle's idea [27] of creating separate queues for the packets from each individual source, generally forwarding packets from different sources in Round Robin fashion, and selectively dropping packets from high-volume sources in order to fairly allocate bandwidth among sources. If a source sends packets too quickly, it makes the length of its own queue grow and more packets in the queue will be dropped. This is because in per-flow queueing, packets belonging to different flows are isolated from each other and one flow cannot have much impact on another. In theoretical perspective, one bit is sent from the flow at a time in Round Robin fashion. In practice, since it is impractical to implement "one-bit-sending", it was suggested to calculate the time when a packet would have left the router using the FQ algorithm and then the packet is inserted into a queue of packets sorted by departure times.

The Fair Queueing (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 managing buffers, processing packet headers or performing packet scheduling on a per-flow basis, 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.) There are several variations of the Fair Queueing scheme [22, 30, 15, 3] but none of them reduces the computation complexity of Fair Queueing[2].

1.4.2 Stochastic Fair Queuing

A proposal called ***Stochastic Fair Queuing*** (SFQ) has been made to economize the computations by hashing which is used to map packets to corresponding queues,

[2] 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.

and using fewer queues [26]. Normally, one queue is required for every possible flow through the router. But based on the assumption that at any particular time the number of active flows are much less than the total number of possible flows through the router, SFQ doesn't have a separate queue for every flow, and flows that hash into the same bucket are treated equivalently. As a result, those flows are treated unfairly. Since many queues (perhaps thousands, but less than the number of queues needed by Fair Queueing) will still be required to achieve high fairness in this method [25], the essential difficulty persists. (Another merit of SFQ is "buffer stealing scheme" — when the buffer is full the packet from the longest queue is dropped — which allows better buffer utilization as buffers are essentially shared by all flows.)

1.4.3 Deficit Round Robin

The ***Deficit Round Robin*** (DRR) algorithm [32] represents an efficient implementation of the well-known ***Weighted Fair Queueing*** (WFQ) discipline[3]. Stochastic fair queuing is used to assign flows to queues, and a slightly-modified round robin scheme with a "credit" of service assigned to each queue is used between the queues — if a particular queue doesn't send any data during a particular round because its packet size is too large, then that "credit" will be built for the next round. But DDR has the same problems as SFQ (a need for many queues).

1.4.4 Core-Stateless Fair Queueing

To address this issue, ***Core-Stateless Fair Queueing*** (CSFQ) was proposed in [33] for achieving reasonably fair bandwidth allocations while reducing the cost. It works by establishing an island of routers (a contiguous region of the network) that implement the protocol. The core of the island ("core" routers of the network) adopt a protocol that does not maintain "state" (such as a record of the volume) for each flow, thereby allowing itself to employ a fast, simple queueing algorithm. However,

[3]WFQ is one of Cisco's premier queuing techniques. It is a flow-based queuing algorithm that does two things simultaneously: It schedules interactive traffic to the front of the queue to reduce response time, and it fairly shares the remaining bandwidth between high bandwidth flows.

such per-flow states still need to be maintained at the borders of the island ("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. Edge routers estimate flows' arrival rates by exponential averaging based on per flow information and insert them into the packet labels; the flows' arrival rates are updated at each router along the path based only on aggregate information at that router. (Another key aspect of the architecture is FIFO queueing with probabilistic dropping: the probability of dropping an arriving packet is a function of the rate estimate carried in the label and of an estimate of the fair share rate at that router.)

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.

1.4.5 Early Random Drop and Random Early Detection

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 ***Early Random Drop*** (ERD) [18] and ***Random Early Detection*** (RED) [11] packets are dropped at random when the queue lengthens, which provides early congestion indication to flows which can then gracefully "backoff" before the buffer is overloaded. RED maintains two buffer thresholds. When the buffer occupancy is smaller than the first threshold, no packet is dropped; when the buffer occupancy is larger than the second threshold all packets are dropped; when the buffer occupancy is between the two thresholds, the packet dropping probability increases linearly with buffer occupancy. This algorithm greatly alleviates the problem of segregation. By keeping the average buffer occupancy small, it reduces the delays of most packets. However, it is now generally agreed that the "RED" algorithm does not provide fair bandwidth allocation and is vulnerable to ill-behaved flows because of its random dropping. It is clear that "ERD" and 'RED" are computationally very easy to implement but do not approximate fairness or achieve objective (2) (*Fairness*).

1.4.6 Flow Random Early Drop

In [23] a modification of these methods, ***Flow Random Early Drop*** (FRED), was proposed in which packets are dropped with probabilities that depend on the flow volumes of their sources, in order to enforce roughly fair bandwidths. 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 active flow subjected this proposal to similar criticism of excessive computational overhead.

1.4.7 Stabilized RED

In the ***Stabilized RED*** (SRED) proposal [28], a conceptively simple method was suggested to estimate the number of active connections, identify high-volume flows, and penalize them by preferentially dropping their packets. However, SRED incurs significantly greater implementation overhead than RED. Other variants of RED (like ***RED with penalty box*** [8] and ***RED with Preferential Dropping*** (RED-PD) [24]) also need to keep certain type of flow information and are complex.

1.4.8 CHOKe

The idea of SRED was further simplified, both from a conceptual and computational point of view, in the ***CHOKe*** proposal [29]. 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, else the randomly selected packet is left intact (in the same position as before) and the incoming packet is dropped or retained based on the same strategy as RED. This has the merit of preferentially 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. CHOKe has been analyzed in [34][35] and it was shown that in the simple case of a single link with homogeneous TCP flows and a single UDP flow, the UDP bandwidth share is at most 26.9 percent of the link capacity when its arrival rate at the router is slightly greater than the link capacity and goes down to zero when the rate increases. Extensive simulations have provided support for its favorable performance with regard to both objectives (1) (*Efficiency*) and (2) (*Fairness*) above, *provided there is just one UDP (unresponsive) flow and all other flows are TCP compliant* [34].

But this result haven't been generalized to multiple links, or even a single link with more than one UDP flow. Simulation results [29][14] have shown that CHOKe performs poorly in the presence of multiple links and/or multiple irresponsive sources. When several flows are unresponsive, it's easy to see that CHOKe will fail to prevent them 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) (*Efficiency*).

1.5 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 multi-player 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) (*Efficiency*) and (2) (*Fairness*).

In this thesis, we will propose three protocols, all of which are based on game theory. The key aspect of our protocol is that by sending no packets from high-volume flows, it gives these flows an incentive to use a more responsive protocol, as the severity of the punishment increases the likelihood that these flows will adapt a more responsive protocol.

We begin now by specifying the technical properties of our protocol. There are two types of properties: (A) Computational properties, (B) Game-theoretic properties. In addition, our third protocol, that we call protocol ***FBA***, has some special properties. We will present them in section 1.5.3.

1.5.1 (A) Computational properties

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

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

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

1.5.2 (B) Game-theoretic properties

Some explanation is needed before presenting the game-theoretic properties. 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 strategy, 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 system, 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 S_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 properties are:

B1. Assume an idealized situation in which the router, and all the sources, know the flow arrival rates $\{s_i\}$; and in which the queue buffer is unbounded. This idealized game has a unique Nash equilibrium which is the "max-min-fairness" rates S_i, given 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 flow arriving 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 in section 4.2:

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.

Next, 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 [34]) 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).

Finally, we will show, in section 5, that property B1 can be extended to the general network case with many routers:

B5. Under the same assumptions for property B1, the game has a unique Nash equilibrium which is the "max-min-fairness" rates R_i.

1.5.3 (C) Special properties of protocol FBA

Since protocol FBA uses an feedback-based adaptive controller to estimate the "max-min-fairness" threshold, there are some special properties of it.

C1. Assume an idealized situation in which the router knows the arrival rates $\{s_i\}$; and in which the queue buffer is unbounded. In this idealized game, any flow, whose arrival rate is at most the "max-min-fairness" threshold α^*, will never be punished.

C2. By properly setting the parameters of protocol FBA, there will be no buffer overflow and no buffer empty in case of congestion.

As a corollary of (C1) and (C2), when the router knows the arrival rates of flows, and the parameters of protocol FBA are properly set, any flow whose arrival rate is at most the "max-min-fairness" threshold α^* will never be punished.

Finally, we'll attend to the stability of protocol FBA:

C3. When the arrival rates of flows are changed, protocol FBA will respond rapidly to current flow condition, i.e., stability could be reestablished quickly in response to changes in arrival rates.

Chapter 2

Protocols

Our protocol is inspired both by network packet queuing theory and by auction theory. From the network packet queuing theory perspective our protocol is similar to CHOKe. In case of congestion, CHOKe penalize all the sources in proportion to their arrival rate. We instead penalize only the highest rate senders. This ensures that the best thing for a sender in case of congestion is to not be the highest rate flow. So all the senders compete not to be the highest rate flow; this process eliminates the congestion. If we consider CHOKe and our protocol in the setting of the auction of a single item, 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 compete to not be the winner, until the total bids are low enough that the auction is cancelled.

We will actually describe three different versions of the protocol. We will begin with protocol I, to which we'll address the theorems of this thesis. Protocol II is very similar, but is better at coping with multiple UDP sources, as will be illustrated by simulation in section 7. The third version, which we call protocol $\boldsymbol{FBA}$, will be given at the end of this section. It is proposed since protocol II has the potential problems of buffer overflow and responding slowly to the fluctuation of flow condition.

2.1 Protocol I

2.1.1 Data structure

The protocol I 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[1] 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.

2.1.2 Actions when a packet arrives

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

1. The packet source i is identified.
2. (a) If $Q > H$, mark the packet *DROP*;
   (b) otherwise if $H \geq Q > L$ and $i = MAX$, mark the packet *DROP*;
   (c) otherwise, mark the packet *SEND*.
3. The packet is appended to the tail of the queue, together with its marking. (If the marking is *DROP*, the packet data can be deleted, but the header is retained so long as the packet is on the queue.)

[1]When congestion is mild (as represented by the length of queue being less than L), a router does not need to regulate the bandwidth of flows.

4. Q and m_i are incremented by 1. (If $m_i = 1$, a new record is created.)

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

2.1.3 Actions when a packet departs

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 marked *SEND*, it is routed it 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.

2.1.4 Comments

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, instead, we use what we call a ***tail-marking*** queue: we make drop decisions at the tail of the buffer, but we don't really drop the packets there; instead, we append the packet to the tail of the buffer together with the marking of *SEND* or *DROP* and send or drop it at the head of the queue according to its marking. 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 the recent traffic volumes from the sources. In our algorithm the contents of the queue represent, in all circumstances, the complete history of packets received at the queue over a recent interval of time. (It is permissible though to only put the header, and not the content of a *DROP* packet on the queue.)

Our handling of drops enables us to use the $\{m_i\}$ 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:

1. *packet-send-time* T_1: the time to route a *SEND* packet at the head of the queue;
2. *packet-append-time* T_2: the time to append a packet to the tail of the queue;
3. *packet-drop-time* T_3: the time to move the head of the queue past a *DROP* packet.

For a queueing system to make sense, these times should satisfy the inequalities $T_1 > T_2 > T_3$, and especially, $T_1 \gg T_3$. 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.)

Comment 3: The marking of *SEND* or *DROP* can be one extra bit to the packet header: the bit is set to one if the marking is *SEND*.

2.2 Protocol II

Protocol II differs from protocol I only in line 2(b) of **Actions when a packet arrives**, which we replace by:

2(b)' If $H \geq Q > L$ and $m_i \geq \frac{H-Q}{H-L} m_{MAX}$, mark the packet *DROP*; otherwise, mark 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 7.

2.3 A Feedback-Based Adaptive Approach (FBA)

Our third protocol is based upon using a current estimate α of the "max-min-fairness" threshold, as a control parameter. We will continually modify α in response to the state of the queue. In this way (as we will analyze in section 6) the length of the queue will remain within acceptable bounds, while we also ensure high throughput and fairness. Although α will fluctuate about its ideal value even under steady flow conditions, we will show through analysis and simulation that these fluctuations are tolerable.

Just as in protocols I and II, we adopt a "penalty" approach: we maintain estimates of the flow rates of the active flows, and drop the packets of those sources that are transmitting above the current threshold determined by the control parameter α. Details in section 2.3.3.

2.3.1 Data structure

Let the maximal possible arrival rate (the maximal number of packets that can be accepted in a unit of time) of the router be S, which is greater than C. Our algorithm will maintain several items of data:

1. α, the estimated "max-min-fairness" threshold.

2. q, the total number of packets, marked *SEND*, presently in the queue.

3. q_{last}, the value of q when α was last updated.

4. A hash table containing, for each flow i having packets in the queue:

(a) m_i, the total number of packets presently in the queue from source i. (Same as in protocol I and II.)

(b) t_i, the time when the last pulled-off packet from source i entered the queue.

In addition, there are several adjustable parameters controlling the algorithm behavior:

1. F, the size of the queue buffer. (Same as protocol I and II.)

2. E, "Equilibrium" marker, satisfying $0 < E < F$.

3. Δt, the time interval between consecutive updates of α.[2]

We will describe the algorithm by two parts: 1. the estimation of "max-min-fairness" threshold; 2. the actions it takes when packets arrive at the router and when they depart the head of the queue.

2.3.2 Estimation of the "max-min-fairness" threshold

A straightforward thought is the following: when the number of packets in the queue marked *SEND* increases, decrement α; when the number of packets in the queue marked *SEND* decreases, increment α. We need a controller to estimate α adaptively according to the arrival traffic of the router.

For easy understanding, we introduce another variable

$$q' := (q - q_{last})/\Delta t,$$

which can be interpreted as a discrete time-derivative of q. So $q = q_{last} + q'\Delta t$.

Let the initial value of α be C. Every Δt time, we will update the value of α once:

1. If $q > E$ and $q' > 0$, then
$$\alpha := \frac{C}{C + q'}\,\alpha;$$

[2]For the question when to update α, here we use a constant interval to update α. In reality, we may update α when a packet is sent out by the router or when a packet arrives at the queue or both; in this case, Δt is interpreted as the time interval between consecutive updates of the estimated threshold.

else if $q < E$ and $q' < 0$, then

$$\alpha := 2\alpha.$$

2. If $\alpha > C$, then $\alpha := C$.

3. $q_{last} := q$.

2.3.2.1 Comments

Comment 1. When $q' > 0$, q increases; when $q' < 0$, q decreases; and when $q' = 0$, q remains unchanged. Since E is the "Equilibrium" marker, if $q > E$ we try to make q less and if $q < E$ we try to make q greater.

Hence, if q is greater than E and will continue increasing ($q' > 0$), the threshold should be decreased; the update equation will reduce the threshold. We will show in Lemma 10 that it will never get a threshold less than α^*, the "max-min-fairness" threshold.

If q is less than E and will continue decreasing ($q' < 0$), the threshold should be increased; the update equation will increase the threshold and ensures quickly getting $q' \geq 0$.

Comment 2. The initial value of the threshold is set to be C because we should not punish flows that are sending at rates less than the "max-min-fairness" threshold and C is the biggest possible value for the threshold, which happens only when there is a single flow in the link.

Comment 3. Our algorithm doesn't mention the case when queue is full or empty. Theorem 14 in Section 6.2 will show that: if $E \geq 2 \cdot \log 2C \cdot C \cdot \Delta t$ and $F \geq (E + \log C \cdot (S + C) \cdot \Delta t) \cdot (2S - C)/C$, the buffer will never overflow or (unless the router is uncongested) be empty.

2.3.3 Actions at the tail and head of the queue

At the tail of the queue, all arrival packets are accepted into the queue, together with a marking *SEND* or *DROP*. For each arriving packet, if its estimated arrival rate is greater than the estimated threshold α, the packet is marked *DROP*, otherwise the packet is marked *SEND*.

At the head of the queue, the packet is sent or dropped according to its marking.

2.3.3.1 Packet arrivals

Each time a packet arrives at the tail of the queue, the following actions are performed:

1. The packet source i is identified.
2. Let the current time be t_{curr}.
3. (a) If $m_i/(t_{curr} - t_i) > \alpha$, mark the packet *DROP*; [3]
   (b) otherwise, mark the packet *SEND*.
4. The packet is appended to the tail of the queue, together with its marking and arriving time t_{curr}.
5. m_i is incremented by 1. (If $m_i = 1$, a new record is created.)
6. If the marking is *SEND*, then q is incremented by 1.

2.3.3.2 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. m_i is decremented by 1. (If $m_i = 0$ the record is eliminated from the hash table.)
3. Set t_i to be the arrival time of the packet.
4. If the packet is marked *SEND*, it is routed it to its destination and q is decremented by 1; otherwise, it is dropped.

[3] $m_i/(t_{curr} - t_i)$ is the estimated arrival rate of flow from source i.

2.3.3.3 Comment

Since we don't "drop-tail", we need to ensure that there is no buffer overflow, which will be shown in Theorem 14 of Section 6.2.

Chapter 3

Protocols I, II, and FBA satisfy the computational properties (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 properties, 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) [13] 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 poly-logarithmic 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 [21] 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.)

Chapter 4

Protocol I satisfies the game-theoretic properties (B)

We assume there are B Poisson sources and their Poisson arrival 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.

In section 4.1, 4.2, and 4.3, we prove the satisfaction of the properties (B) for protocol I.

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 2 *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 $s_i = S_i$, for S_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^*. $\square$

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 3 *X is a Poisson process with $E[X] = \lambda$:*

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

Proof: As in the proof of the Chernoff bound, for $\delta > 0$,

$$X \geq (1+\delta)E[X] \text{ if and only if } e^{tX} \geq e^{t(1+\delta)E[X]}.$$

So by Markov's inequality

$$Pr\{X \geq (1+\delta)E[X]\} \leq \frac{E[e^{tX}]}{e^{t(1+\delta)E[X]}}.$$

Since X is a Poisson process with $E[X] = \lambda$,

$$E[e^{tX}] = \sum_{k \in \mathbb{N}} \frac{e^{tk}e^{-\lambda}\lambda^k}{k!} = e^{-\lambda} \sum_{k \in \mathbb{N}} \frac{\lambda e^{tk}}{k!} = e^{-\lambda}e^{\lambda e^t} = e^{\lambda(e^t-1)}.$$

So

$$Pr\{X \geq (1+\delta)\lambda\} \leq e^{\lambda(e^t-1)}e^{-t(1+\delta)\lambda} = e^{\lambda(e^t-1)-t(1+\delta)\lambda}.$$

To minimize the exponent, substitute the value $\ln(1+\delta)$ for t, and we conclude

$$\begin{aligned} Pr\{X \geq (1+\delta)\lambda\} &\leq e^{(1+\delta)\lambda\ln(1/(1+\delta))+(1+\delta)\lambda-\lambda} \\ &= e^{-((1+\delta)\ln(1+\delta)-\delta)\lambda} = \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\lambda. \end{aligned}$$

When $\delta \geq 2e-1$,

$$Pr\{X \geq (1+\delta)\lambda\} < 2^{-\lambda\delta},$$

and when $\delta < 2e-1$,

$$Pr\{X \geq (1+\delta)\lambda\} < e^{-\lambda\delta^2/4}.$$

Similarly,

$$X \leq (1-\delta)E[X] \text{ if and only if } e^{tX} \geq e^{t(1+\delta)E[X]}$$

for any $\delta \in (0,1]$. By Markov's inequality

$$Pr\{X \leq (1-\delta)E[X]\} \leq \frac{E[e^{-tX}]}{e^{-t(1-\delta)E[X]}}.$$

Since

$$\begin{aligned} E[e^{-tX}] &= \sum_{k\in\mathbb{N}} \frac{e^{-tk}e^{-\lambda}\lambda^k}{k!} = e^{-\lambda}\sum_{k\in\mathbb{N}}\frac{\lambda e^{-tk}}{k!} \\ &= e^{-\lambda}e^{\lambda e^{-t}} = e^{\lambda(e^{-t}-1)}, \end{aligned}$$

so

$$Pr\{X \leq (1-\delta)\lambda\} \leq e^{\lambda(e^{-t}-1)}e^{t(1-\delta)\lambda} = e^{\lambda(e^{-t}-1)+t(1-\delta)\lambda}.$$

Choose $t = -\ln(1-\delta)$, and we conclude

$$\begin{aligned} Pr\{X \le (1-\delta)\lambda\} &\le e^{(1-\delta)\lambda\ln(1/(1-\delta))+(1-\delta)\lambda-\lambda} = e^{-((1-\delta)\ln(1-\delta)+\delta)\lambda} \\ &= \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\lambda} < \left(\frac{e^{-\delta}}{e^{-\delta+\delta^2/2}}\right)^{\lambda} = e^{-\lambda\delta^2/2}. \end{aligned}$$

□

Lemma 4 *X and Y are two Poisson processes with $E[X] = \alpha^*$, $E[Y] = \beta$, and $\alpha^* < \beta$:*

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

Proof: Since $\alpha^* < \beta$, for any $\varepsilon \in (0, \frac{\beta-\alpha^*}{\alpha^*+\beta}]$, we have $\alpha^*(1+\varepsilon) \le \beta(1-\varepsilon)$. So

$$\begin{aligned} Pr\{X \ge Y\} &\le Pr\{X \ge \alpha^*(1+\varepsilon)\}Pr\{Y \ge \beta(1-\varepsilon)\} \\ &\qquad + Pr\{X \ge Y\}Pr\{Y \le \beta(1-\varepsilon)\} \\ &\le Pr\{X \ge \alpha^*(1+\varepsilon)\} + Pr\{Y \le \beta(1-\varepsilon)\} \\ &\le e^{-\alpha^*\varepsilon^2/4} + e^{-\beta\varepsilon^2/2}. \end{aligned}$$

□

The basic argument survives with the hypothesis $L \in \Omega(\widetilde{B}\frac{1}{\varepsilon^2}\ln\frac{1}{\varepsilon})$.

Theorem 5 *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 $0 \le s_i \le r_i$, there is a small $\varepsilon > 0$ such that any source sending at rate $s_i \le (1-\varepsilon)\alpha^*$, will achieve throughput $a_i \ge s_i(1-\varepsilon)$. (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: $\sum_j s_j \ge \sum_j \min\{(1-\varepsilon/2)\alpha^*, r_j\}$.

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 arrival 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

$$\begin{aligned} Pr\{m_i = m_{MAX}\} &\leq Pr\left\{m_i \frac{(1-\delta_0)(1+\delta)}{(1+\delta_0)(1-\delta)} = m_{MAX}\right\} \\ &\leq Pr\left\{m_i \frac{(1-\delta_0)(1+\delta)}{(1+\delta_0)(1-\delta)} \geq m_k\right\} \\ &< e^{-s_i \frac{(1-\delta_0)(1+\delta)}{(1+\delta_0)(1-\delta)} t\delta_0^2/4} + e^{-s_k t \delta_0^2/2} \\ &\leq 2e^{-s_k t(1-\delta_0)\delta_0^2/(4(1+\delta_0))}. \end{aligned}$$

Let $f(\delta) = (1-\delta)\delta^2/(1+\delta)$. Since $L \geq \Omega(\frac{1}{\varepsilon^2}\ln\frac{1}{\varepsilon})\cdot\widetilde{B}$, we can assume $\frac{L}{\widetilde{B}} > 4\ln(\frac{2}{\varepsilon})(\frac{32}{\varepsilon^2} - \frac{32}{\varepsilon} + 10 + \frac{\varepsilon}{1-\varepsilon})$. When $\varepsilon \leq \frac{14-2\sqrt{5}}{11}$, $f(\varepsilon/(4-3\varepsilon)) \leq f(\delta_0)$ and

$$\begin{aligned} L/\widetilde{B} > 8\ln\frac{2}{\varepsilon}/f(\varepsilon/(4-3\varepsilon)) &\Rightarrow L/\widetilde{B} > 8\ln\frac{2}{\varepsilon}/f(\delta_0) \\ &\Rightarrow (s_k L)/(2s_k\widetilde{B}) > 4\ln(\frac{2}{\varepsilon})/f(\delta_0). \end{aligned}$$

Because $t \approx L/\sum_{i=1}^{B} s_i \geq \frac{1}{2}L/\sum_{i=1}^{\widetilde{B}} s_i \geq L/(2\widetilde{B}s_k)$, so $s_k t > 4\ln(\frac{2}{\varepsilon})/f(\delta_0)$, and then $Pr\{m_i = m_{MAX}\} \leq 2e^{-s_k t f(\delta_0)/4} < \varepsilon$. 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.

Consider a period $[0,t]$: at time 0 the queue is empty and it is never empty again from time 0 to t. Since the sources are constant-rate Poisson and independent, the number of arrival packets, M, follows Poisson distribution with rate $\sum_j s_j$, whose expectation is $\sum_j s_j \cdot t$. And $Q = M - C \cdot t$ is the number of packets in the queue at time t. By Lemma 3,

$$Pr\{Q > L\} < Pr\{M > C \cdot t + L\} < \begin{cases} e^{-(\sum_j s_j)\cdot t(c-1)^2/4} & , \quad c \in (1, 2e) \\ 2^{-(\sum_j s_j)\cdot t(c-1)} & , \quad c \in [2e, +\infty) \end{cases},$$

where $c = C/\sum_j s_j + L/(\sum_j s_j \cdot t)$.

(a) If $C/\sum_j s_j > 2$ or $L/(\sum_j s_j \cdot t) > 1$, then

$$Pr\{Q > L\} < 2^{-(\sum_j s_j)\cdot t(c-1)/4}.$$

When $L > 4 \log_2 \frac{1}{\varepsilon}$,

$$Pr\{Q > L\} < \varepsilon.$$

(b) If $C/\sum_j s_j < 2$ and $L/(\sum_j s_j \cdot t) < 1$, then

$$Pr\{Q > L\} < e^{-(\sum_j s_j)\cdot t(c-1)^2/4}.$$

Let B' be the number of sources whose $r_i \geq \alpha^*$, so $\sum\limits_j s_j \leq C - B'\alpha^*\varepsilon/2$.
When $L > 4(\frac{2C}{B'\alpha^*\varepsilon} - 1)^2 \ln \frac{1}{\varepsilon}$,

$$e^{-(\sum_j s_j)\cdot t(c-1)^2/4} \leq e^{-L(c-1)^2/4} \leq e^{-L(\frac{c}{\sum_j s_j}-1)^2/4} \leq e^{4\log\varepsilon/4}$$
$$\Rightarrow \quad Pr\{Q > L\} < \varepsilon.$$

Since $L \geq \Omega(\frac{1}{\varepsilon^2} \ln \frac{1}{\varepsilon}) \cdot \widetilde{B}$, the probability that the length of the queue is larger than L is less than ε and source i will achieve throughput $a_i \geq s_i(1-\varepsilon)$. $\square$

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 $\mathcal{A}$) are fixed. We show that any varying behavior of source $\mathcal{A}$ is no better than a constant transmission rate.

Theorem 6 *For a source which is allowed to send packets at arbitrary times (while all other sources are still restricted to being Poisson), the long-term throughput is no more than $1 + \varepsilon$ times that of the best Poisson strategy.*

Proof: Assume source $\mathcal{A}$ 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 (with $s_1 \geq s_2 \geq \cdots \geq s_B$).

Consider the moment t when a packet from source $\mathcal{A}$ comes and we assume that the arrival time of the packet at the head of the queue is t'. If the marking together with the packet is *SEND*, it means that $\mathcal{A} \neq MAX$, so in the time interval $[t', t]$, the number of packets from source $\mathcal{A}$ is not the maximal; otherwise, the marking will be *DROP*.

Consider the packets from source $\mathcal{A}$ arriving in $[T, T']$. We ignore the last a few packets which are dropped as they have no contribution to the throughput. Look on the last packet p_1 from source $\mathcal{A}$ which is not dropped, and assume its arrival time is t_1. Suppose at time t_1, the arrival time of the packet at the head of the queue is t'_1. Clearly, in $[t'_1, t_1]$, the number of packets coming from source $\mathcal{A}$ is not the maximal, i.e., $m_{\mathcal{A}} \leq m_{MAX}$. So in $[t'_1, T']$, the number of routed packets of source $\mathcal{A}$ is no more than m_{MAX} at time t_1. Considering the packets arriving before t'_1, we can, similarly, define t_2 as the arrival time of the last packet p_2(arriving before t'_1), which is not dropped, and t'_2 as the arrival time of the packet at the head of the queue at t_2. So in the interval $[t'_2, t'_1]$, the number of routed packets of source $\mathcal{A}$ is no more than m_{MAX} at time t_2. By the same argument, we can split the interval $[T, T']$ into $n+1$ sub-intervals: $[T, t'_n]$, $[t'_n, t'_{n-1}]$, $[t'_{n-1}, t'_{n-2}]$, $\cdots$, $[t'_2, t'_1]$, $[t'_1, T']$. Suppose that $t'_0 = T'$

and $m_{MAX} = MAX_i$ at time t'_i ($i \in \{0, 1, \cdots, n-1\}$). Hence, the number of routed packets of source $\mathcal{A}$ in $[t'_n, T']$, N, is no more than $MAX_0(t'_0 - t'_1) + MAX_1(t'_1 - t'_2) + MAX_2(t'_2 - t'_3) + \cdots + MAX_{n-1}(t'_{n-1} - t'_n)$. Given that $[T, T']$ is long enough, the throughput of source $\mathcal{A}$ in $[T, T']$ is approximate to that of source $\mathcal{A}$ in $[t'_n, T']$.

Let $f(k)$ be the probability that m_{MAX} is at least k, i.e., $f(k) = Pr\{m_{MAX} \geq k\}$. For any two sources i, j of $1, 2, 3, \cdots, B$, if $s_i + s_j \leq s_1$, look at the two sources as one with rate $s_i + s_j$; repeat the procedure until we cannot find such two sources, at which time there is at most one source whose rate is no more than $s_1/2$. Suppose the sources are $1', 2', 3', \cdots, B'$ with rates $s_{1'}, s_{2'}, \cdots, s_{B'}(s_{1'} \geq s_{2'} \geq \cdots \geq s_{B'-1} > s_1/2)$ now. Since $\widetilde{B}s_1 \geq \frac{1}{2}\sum_{i=1}^{B} s_i = \frac{1}{2}\sum_{i=1}^{B'} s_{i'} > \frac{1}{2}\left(s_{B'} + \sum_{i=1}^{B'-1}\frac{s_1}{2}\right) > \frac{1}{4}(B'-1)s_1$, we have $B' < 4\widetilde{B} + 1$. Since $Pr\{m_{MAX} \geq k\} = Pr\{\exists i \in [1, B], m_i \geq k\} \leq Pr\{\exists i, j \in [1, B], m_i + m_j \geq k\}$, $f(k) \leq \min\{\sum_{i=1}^{B'} Pr\{m_{i'} \geq k\}, 1\} < \min\{(4\widetilde{B}+1)Pr\{m_1 \geq k\}, 1\}$. So in any interval $[t'_{i+1}, t'_i]$,

$$
\begin{aligned}
& E[MAX_i] \\
= \; & \int_0^\infty f(k)dk = \int_0^{s_1(t'_i - t'_{i+1})(1+\delta)} f(k)dk + \int_{s_1(t'_i - t'_{i+1})(1+\delta)}^\infty f(k)dk \\
\leq \; & s_1(t'_i - t'_{i+1})(1+\delta) + \int_{s_1(t'_i - t'_{i+1})(1+\delta)}^\infty (4\widetilde{B}+1)Pr\{m_1 \geq k\}dk \\
\leq \; & s_1(t'_i - t'_{i+1})(1+\delta) + (4\widetilde{B}+1)\int_{s_1(t'_i - t'_{i+1})(1+\delta)}^\infty e^{-s_1(t'_i - t'_{i+1})(\frac{k}{s_1(t'_i - t'_{i+1})}-1)^2/4}dk \\
\leq \; & s_1(t'_i - t'_{i+1})(1+\delta) + (4\widetilde{B}+1)s_1(t'_i - t'_{i+1})\int_\delta^\infty e^{-s_1(t'_i - t'_{i+1})x^2/4}dx \\
\leq \; & s_1(t'_i - t'_{i+1})(1+\delta) + 2(4\widetilde{B}+1)\sqrt{\pi s_1(t'_i - t'_{i+1})}\int_{\frac{\sqrt{s_1(t'_i - t'_{i+1})}\delta}{\sqrt{2}}}^\infty \frac{1}{\sqrt{2\pi}}e^{-x^2/2}dx \\
\leq \; & s_1(t'_i - t'_{i+1})(1+\delta) + 2(4\widetilde{B}+1)\frac{\sqrt{\pi s_1(t'_i - t'_{i+1})}}{\frac{\sqrt{s_1(t'_i - t'_{i+1})}\delta}{\sqrt{2}}\sqrt{2\pi}}\, e^{-(\frac{\sqrt{s_1(t'_i - t'_{i+1})}\delta}{\sqrt{2}})^2/2} \\
= \; & s_1(t'_i - t'_{i+1})(1+\delta) + \frac{2(4\widetilde{B}+1)}{\delta}\, e^{-\frac{s_1(t'_i - t'_{i+1})\delta^2}{4}}.
\end{aligned}
$$

Since $t'_i - t'_{i+1} \geq L/(2\widetilde{B}s_1)$,

$$E[MAX_i] \leq s_1(t'_i - t'_{i+1})(1+\delta) + \frac{2(4\widetilde{B}+1)}{\delta}\, e^{-\frac{L\delta^2}{8\widetilde{B}}}.$$

Under the assumption that $L \geq \frac{8\widetilde{B}}{\delta^2} \ln \frac{2(4\widetilde{B}+1)}{\delta^2}$ ($L = \Omega(\frac{1}{\varepsilon^2} \ln \frac{1}{\varepsilon}) \cdot \widetilde{B}$), $\frac{2(4\widetilde{B}+1)}{\delta} e^{-\frac{L\delta^2}{8\widetilde{B}}} \leq \delta$. Let $\delta = \varepsilon/2$, and then

$$E[MAX_i] \leq s_1(t'_i - t'_{i+1})(1+\varepsilon/2) + \varepsilon/2 \leq s_1(t'_i - t'_{i+1})(1+\varepsilon).$$

So in the interval $[T, T']$, the expected number of routed packets of source $\mathcal{A}$ is no more than $1+\varepsilon$ times that of arrival packets from source 1, which is $s_1(T'-T)$. The best throughput for source $\mathcal{A}$, whose desired rate is greater than s_1, is no more than $1+\varepsilon$ times that of the Poisson strategy with rate s_1. □

Hence, it is to the advantage of a source to sending packets at "max-min-fairness" rate. While all the sources, except one source $\mathcal{A}$, are sending packets obeying Poisson process with their "max-min-fairness" rates, the best strategy for source $\mathcal{A}$ is to be Poisson process with his "max-min-fairness" rate.

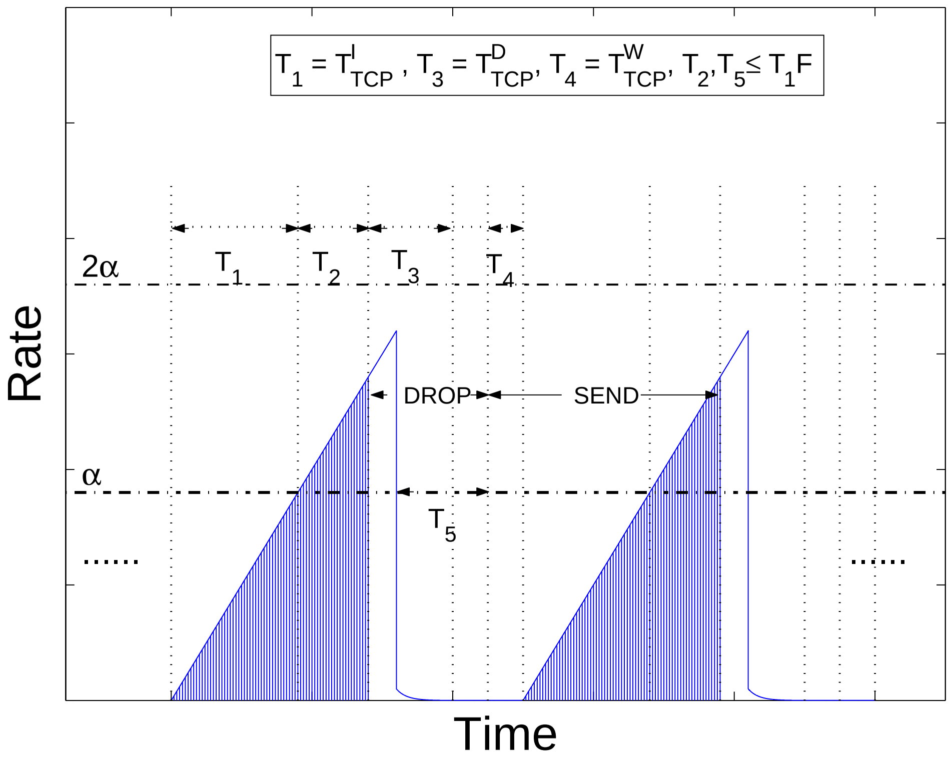
4.3 Performance of TCP: property B4

We give here a brief overview of TCP from the perspective of the theory community. 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 source) 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 "max-min-fairness" threshold α^*, is not precisely known, we assume that 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 $2\alpha^*$, 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 $2\alpha^*$ 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 4.1.

Theorem 7 *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 $2\alpha^*$, 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 4.1, 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 $D \leq 8$. $\square$



$T_1 = T^I_{TCP}$, $T_3 = T^D_{TCP}$, $T_4 = T^W_{TCP}$, T_2,$T_5 \leq T_1F$
2α
α
T_1
T_2
T_3
T_4
T_5
DROP
SEND
......
......
Rate
Time

Figure 4.1: 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.

Chapter 5

Protocol I satisfies the network equilibrium property (B5)

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 [20] 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.)

The final throughput of each flow, in a network, is decided by the router on which it has the lowest throughput. We can find the "max-min-fairness" rate of each flow in the network by the following steps:

1. Look on each router in the network as disconnected. On each router, there is a unique Nash equilibrium, i.e., a unique "max-min-fairness" rate for each flow on the router.

2. Repeat the following steps until there is no flow left:

   (a) Compute the "max-min-fairness" rate for each flow on every router.

   (b) Among all the rates we get, pick a lowest one, which belongs to a flow i, and set it to be the "max-min-fairness" rate, S_i, of flow i.

   (c) Delete the flow i and decrease the capacity of those routers, which flow i goes through, by S_i.

Finally, we will get a "max-min-fairness" rate for each flow. Since every time we choose the lowest one (if there are several, choose one of them), which is the best throughput the flow may get, it is to the advantage of the flow to send packets at that rate as it is the most efficient way and the lost is the smallest.

This reduces our task to showing:

Lemma 8 *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 only 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 $\mathcal{A} \in \{1, \ldots, \mathcal{N}\}$(where $\mathcal{N}$ is the number of the flows) such that $y_{\mathcal{A}} > x_{\mathcal{A}}$, then $x_{\mathcal{A}} < r_{\mathcal{A}}$ (where $r_{\mathcal{A}}$ is the desired rate of flow

$\mathcal{A}$). Hence, there exists a router A carrying flow $\mathcal{A}$ and such that the bandwidth of any other flow on A is at most $x_{\mathcal{A}}$. (Otherwise, for any router, $x_{\mathcal{A}}$ is not the highest rate it carries, so $x_{\mathcal{A}}$ can be increased and $\overrightarrow{x}$ is not a Nash equilibrium.) So there exists a $t \in \{1, \ldots, \mathcal{N}\}$, $t \neq \mathcal{A}$, such that $y_t < x_t \leq x_{\mathcal{A}}$. 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[1]. Under this assumption we have:

Theorem 9 *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.*

[1]If a sender is sending at rate s and its "max-min-fairness" rate is r, since the "max-min-fairness" rate of a flow is the best throughput it may get, the best gain of the sender is $r - \varepsilon(s-r) = (1+\varepsilon)r - \varepsilon s$. In order to gain as much as possible, the best bet for the sender is to send packets at its "max-min-fairness" rate. It is reasonable to assume that there is a cost per packet loss, because sending at a higher rate means more loss of packets and then the sender needs to use an error-correcting code and take more processing time to send packets. Therefore, the competing senders will choose the Nash equilibrium which is the "max-min-fairness" allocation.

Chapter 6

Protocol FBA satisfies properties (C)

In this section, we still use C to represent router capacity and S to be the maximal possible arrival rate of the router. Moreover, for each flow i, let its arrival rate be ${s_i}$[1]. Assume the "max-min-fairness" threshold to be α^*. Because of the properties of router and network, α^* always has a lower bound[2]; here, we assume that $\alpha^* \geq 1$.

We will first prove the satisfaction of property C1 in section 6.1. Then we show the bounded queue length property C2 of the protocol in section 6.2. Together with property C1, this shows the fairness of the protocol. Finally, we will analyze protocol FBA for its efficiency and stability (property C3) properties.

6.1 Fairness in idealized situation: property C1

As indicated in section 1.5.3, an idealized situation is where the router knows the source rates $\{s_i\}$ and the queue buffer is unbounded. Moreover, the source rates are fixed. In this section, we will assume that our game is under an idealized situation.

First, we rewrite the update formula of α as follows:

[1]We are using the exact value here. But in the actual algorithm, we don't know the exact arrival rate and we use an estimate $m_i/(t_{curr} - t_i)$ instead.

[2]For performance of the network, the delay over the router should not be too big; the time that a packet stays in the queue of router has an upper bound. If we have a record for the flow, it has at least 1 packet in the queue. Thus, the flow's arrival rate is lower bounded.

1. If $q_k > E$ and $q'_k > 0$, then

$$\alpha_{k+1} = \frac{C}{C + q'_k}\ \alpha_k;$$

   else if $q_k < E$ and $q'_k < 0$, then

$$\alpha_{k+1} = 2\alpha_k.$$

2. If $\alpha_{k+1} > C$, then $\quad \alpha_{k+1} = C$.

3. $q_{k+1} = q_k + q'_k \cdot \Delta t$.

The fluctuation of the value of q comes from two reasons: packets arrive at the tail of the queue, marked *SEND*, and packets are sent out at the head of the queue. At each time, approximately C packets[3] are sent out at the head of the queue, so

$$q'_k \approx \sum_{i:s_i \le \alpha_k} s_i - C.$$

It's clear that

$$-C \le q'_k \le S. \tag{6.1}$$

Lemma 10 *In an idealized situation, for any value α_k of the estimated threshold, the α_{k+1} we get from the adaptive approach will never be less than α^*, the "max-min-fairness" threshold.*

Proof: Since when $q \le E$ we won't decrease α_k, we only need to consider the case when $q > E$.

Suppose the arrival rates of flows are s_1, s_2, $\cdots$, s_n, and $s_1 \le s_2 \le \cdots \le s_n$. Since

[3]Since there are some packets dropped at the head of the queue, the actual number may be C minus a small ε. But as the packet-send-time is much greater than the packet-drop-time ($T_1 \gg T_3$), the ε is pretty small compared with C. We will omit this ε in the sequel.

the "max-min-fairness" threshold is α^*, we have that

$$\sum_{i=1}^{n} \min\{s_i, \alpha^*\} = C.$$

There exists a number j with the following properties:

$$\sum_{i=1}^{j-1} s_i \le C \quad \text{and} \quad \sum_{i=1}^{j} s_i > C.$$

Obviously, $\alpha^* < s_j$.

For any value α_k of the threshold, if $\alpha_k < s_j$, then $q'_k \le 0$ and α will not be decremented any more. If $\alpha_k \ge s_j$, then $q'_k > 0$ and we need to decrement α using the formula $\alpha_{k+1} = \frac{C}{C+q'_k} \cdot \alpha_k$. Let l be the biggest number with $s_l \le \alpha_k$. So

$$q'_k \approx \left(\sum_{i:s_i \le \alpha_k} s_i \right) - C = \left(\sum_{i=1}^{l} s_i \right) - C,$$

and $l \ge j$. Introduce a new variable $\widetilde{\alpha}$ with $\sum\limits_{i=1}^{l} \min\{s_i, \widetilde{\alpha}\} = C$, so $\widetilde{\alpha} \ge \alpha^*$ and

$$q'_k \approx \sum_{i=1}^{l} (s_i - \min\{s_i, \widetilde{\alpha}\}).$$

Hence,

$$\frac{q'_k}{C} \approx \frac{\sum\limits_{i=1}^{l} (s_i - \min\{s_i, \widetilde{\alpha}\})}{\sum\limits_{i=1}^{l} \min\{s_i, \widetilde{\alpha}\}} \le \frac{s_l - \widetilde{\alpha}}{\widetilde{\alpha}} \le \frac{\alpha_k - \alpha^*}{\alpha^*},$$

and then

$$\begin{aligned} & \frac{\alpha_{k+1}}{\alpha_k} = \frac{C}{C + q'_k} = \frac{1}{1 + \frac{q'_k}{C}} \ge \frac{1}{1 + \frac{\alpha_k - \alpha^*}{\alpha^*}} = \frac{\alpha^*}{\alpha_k} \\ \Rightarrow \quad & \alpha_{k+1} \ge \alpha^* \end{aligned}$$

□

As a corollary, we have the following theorem:

Theorem 11 *If the source rates $\{s_i\}$ are unchanging and known to the router, and the queue buffer is unbounded, we will never drop any packet from a flow whose arrival rate is at most the "max-min-fairness" threshold α^*.*

6.2 Queue length properties: property C2

We will give upper and lower bounds on q in this section. Here, we still assume that the source rates $\{s_i\}$ are unchanging and known to the router.

Lemma 12 *Under the situation that the source rates $\{s_i\}$ are unchanging and known to the router, q is at most $E + \log C \cdot (S + C) \cdot \Delta t$, and if $F \geq (E + \log C \cdot (S + C) \cdot \Delta t) \cdot (2S - C)/C$, there will be no buffer overflow.*

Proof: Let's first count the maximal number q_{max} of packets that could have *SEND* marking in the queue, which is achieved when q stops increasing ($q > E$), i.e., when $q' \leq 0$. When $q > E$ but $q' > 0$,

$$\alpha_{k+1} = \frac{C}{C + q'_k}\,\alpha_k \Rightarrow \frac{\alpha_{k+1}}{\alpha_k} = \frac{C}{C + q'_k} \quad (\alpha_{k+1} < \alpha_k).$$

So

$$\begin{aligned}
\log \frac{\alpha_{k+1}}{\alpha_k} = \log \frac{C}{C + q'_k} \quad & \Rightarrow \quad \log \alpha_{k+1} - \log \alpha_k = \log(1 - \frac{q'_k}{C + q'_k}) \\
& \Rightarrow \quad \log \alpha_{k+1} - \log \alpha_k \leq -\frac{q'_k}{C + q'_k} \\
& \Rightarrow \quad \log \alpha_k - \log \alpha_{k+1} \geq \frac{q'_k}{C + q'_k} \\
& \overset{(a)}{\Rightarrow} \quad \log \alpha_k - \log \alpha_{k+1} \geq \frac{q'_k}{S + C} \\
& \Rightarrow \quad (\log \alpha_k - \log \alpha_{k+1}) \cdot (S + C) \geq q'_k \\
& \Rightarrow \quad (\log \alpha_k - \log \alpha_{k+1}) \cdot (S + C) \cdot \Delta t \geq q'_k \cdot \Delta t,
\end{aligned}$$

where (a) follows from inequality (6.1).

The increment of q while the threshold is increasing from α_k to α_{k+1} is $q'_k \Delta t$. Let α_0 be the threshold value when q first becomes greater than E and α_{l+1} be the first α with $q' \leq 0$. So q_{max} is

$$\begin{aligned} q_{max} &= E + \sum_{k=0}^{l} q'_k \cdot \Delta t \leq E + \sum_{k=0}^{l} (\log \alpha_k - \log \alpha_{k+1}) \cdot (S + C) \cdot \Delta t \\ &= E + (\log \alpha_0 - \log \alpha_{l+1}) \cdot (S + C) \cdot \Delta t \\ &\overset{(a)}{\leq} E + (\log C - \log \alpha^*) \cdot (S + C) \cdot \Delta t \\ &\overset{(b)}{\leq} E + \log C \cdot (S + C) \cdot \Delta t, \end{aligned}$$

where

(a) follows from the algorithm ($\alpha_0 \leq C$) and Lemma 10 ($\alpha_{l+1} \geq \alpha^*$).

(b) follows from the assumption $\alpha^* \geq 1$.

Let the number of packets in the queue be Q. When q reaches q_{max}, the number of packets in the queue, Q_0, satisfies the inequality $\frac{q_{max}}{C} \geq \frac{Q_0}{S}$. Q will stop increasing before or when these q_{max} packets marked *SEND* have been sent. For every C packets sent out, Q will increase by at most $S - C$, so

$$\frac{Q_{max} - Q_0}{S - C} \leq \frac{q_{max}}{C}.$$

Thus,

$$Q_{max} \leq \frac{q_{max}}{C} (2S - C).$$

Therefore, if

$$F \geq (E + \log C \cdot (S + C) \cdot \Delta t) \cdot (2S - C)/C,$$

there will be no buffer overflow. □

Lemma 13 *Assuming that source rates $\{s_i\}$ are unchanging and known to the router, under steady-state congested conditions, after the first time q exceeds E, q is at least $E - 2 \log 2C \cdot C \cdot \Delta t$.*

Proof: We will use a method similar to that used in Lemma 12. When q becomes less than E, q will keep decreasing until $q' \geq 0$. When $q < E$ but $q' < 0$,

$$\alpha_{k+1} = 2\alpha_k \Rightarrow \frac{\alpha_{k+1}}{\alpha_k} = 2 \quad (\alpha_{k+1} > \alpha_k).$$

So

$$\begin{aligned}
& \frac{\alpha_{k+1}}{\alpha_k} = 2 \overset{(a)}{\geq} \frac{2C}{2C + q'_k} \\
\Rightarrow \quad & \log \frac{\alpha_{k+1}}{\alpha_k} \geq \log \frac{2C}{2C + q'_k} = \log \frac{1}{1 + \frac{q'_k}{2C}} = -\log(1 + \frac{q'_k}{2C}) \geq -\frac{q'_k}{2C} \\
\Rightarrow \quad & (\log \alpha_{k+1} - \log \alpha_k) \cdot 2C \cdot \Delta t \geq -q'_k \cdot \Delta t.
\end{aligned}$$

where (a) follows from inequality (6.1).

The decrement of q between consecutive updates of α (from α_k to α_{k+1}) is $-q'_k \Delta t$. Let α_0 be the threshold value when q becomes less than E and α_{l+1} be the first α with $q' \geq 0$. So the total decrement of q is

$$\begin{aligned}
\sum_{i=0}^{l} -q'_k \cdot \Delta t \;\; & \leq \;\; \sum_{i=0}^{l} (\log \alpha_{k+1} - \log \alpha_k) \cdot 2C \cdot \Delta t \;=\; (\log \alpha_{l+1} - \log \alpha_0) \cdot 2C \cdot \Delta t \\
& \overset{(a)}{\leq} \;\; (\log 2C - \log \alpha^*) \cdot 2C \cdot \Delta t \overset{(b)}{\leq} 2 \log 2C \cdot C \cdot \Delta t,
\end{aligned}$$

where

(a) follows from Lemma 10($\alpha_0 \geq \alpha^*$) ; $\alpha_{l+1} \leq 2C$ because $\alpha_l \leq C$.

(b) follows from the assumption $\alpha^* \geq 1$.

So under steady-state congested conditions, after the first time q exceed E, q is at least $E - 2\log 2C \cdot C \cdot \Delta t$. Therefore, if $E \geq 2 \log 2C \cdot C \cdot \Delta t$, there are packets marked *SEND* in the queue at all time in case of congestion. □

Combining Lemma 12 and Lemma 13, we have the following theorem:

Theorem 14 *If the source rates $\{s_i\}$ are unchanging and known to the router, $E \geq 2 \log 2C \cdot C \cdot \Delta t$, and $F \geq (E + \log C \cdot (S + C) \cdot \Delta t) \cdot (2S - C)/C$, there will be no* *buffer overflow and no buffer empty in case of congestion.*

From Lemma 10, we know that protocol FBA will never get an α less than the "max-min-fairness" threshold α^* under idealized situation. Together with Theorem 14, we have the following theorem for the fairness of protocol FBA:

Theorem 15 *If the source rates $\{s_i\}$ are unchanging and known to the router, $E \geq 2\log 2C \cdot C \cdot \Delta t$, and $F \geq (E + \log C \cdot (S+C) \cdot \Delta t) \cdot (2S - C)/C$, protocol FBA will never drop any packet from a flow whose arrival rate is less than the "max-min-fairness" threshold α^*.*

6.3 Efficiency

Besides fairness, we also need that the router will always send packets near its capacity C. Efficiency can be achieved as long as the following three requirements are satisfied:

1. There are packets marked *SEND* in the queue.
2. The packet-drop-time is short compared with the packet-send-time.
3. The per-packet time complexity of implementing the router congestion control is constant.

From Lemma 13, with $E \geq 2\log 2C \cdot C \cdot \Delta t$, there are packets marked *SEND* in the queue at all time in case of congestion. So requirement 1 is satisfied.

In our queue, the packet-send-time T_1 is much greater than the packet-drop-time T_3: $T_1 \gg T_3$. So requirement 2 is also satisfied.

In section 3, we have discussed the per-packet time complexity for implementing the router congestion control. In order to get constant-time complexity per packet, we suggest that instead of updating m_i and t_i with every packet, we perform these updates only every $(\log F)$'th packet. By doing this, our average per-packet operating time is constant. The favorable properties of the protocol will hold with a slight loss in quality due to the slightly less accurate estimation of flow rates, and the slightly slower reactions to changes in flow rates. More details are given in section 3.

Therefore, with our approach, the router will always send packets near its capacity C.

6.4 Stability: property C3

The stability concept in our approach has a slightly different meaning from the general "stability" concept in control theory. In our approach, we don't require the system to stabilize at an equilibrium point or in a small area around the equilibrium point. Our stability means that *the estimated value of α is always not less than the "max-min-fairness" threshold; there is no buffer overflow; and as long as the router is congested, the queue will not become empty.* If all those requirements are satisfied, the system is in the **good region**. By Theorem 14 and Theorem 15, our algorithm is shown to be "stable" when the arrival rates of flows are fixed and known to the router.

Strictly, when the arrival rates of flows are fixed and known to the router, the ideal value of our estimate α is greater than the "max-min-fairness" threshold in case of congestion. The estimate α fluctuates about its ideal value, and will never below the "max-min-fairness" threshold α^*. Suppose there are n flows, and the arrival rates of flows are s_1, s_2, $\cdots$, s_n, with $s_1 \leq s_2 \leq \cdots \leq s_n$, $\sum_i s_i > C$. We have that

$$\sum_{i=1}^{n} \min\{s_i, \alpha^*\} = C.$$

Also, there exists a number j with the following properties:

$$\sum_{i=1}^{j-1} s_i \leq C \quad \text{and} \quad \sum_{i=1}^{j} s_i > C.$$

Obviously, $\alpha^* < s_j$. From our update formula of α, we know that s_j is the ideal value of our estimate α. Figure 6.1 shows the relationship between the "max-min-fairness" threshold α^* and the ideal value s_j of our estimate α.

When the arrival rates of flows change, the "max-min-fairness" threshold α^* of the system changes accordingly, and then the system will need some time to stabilize

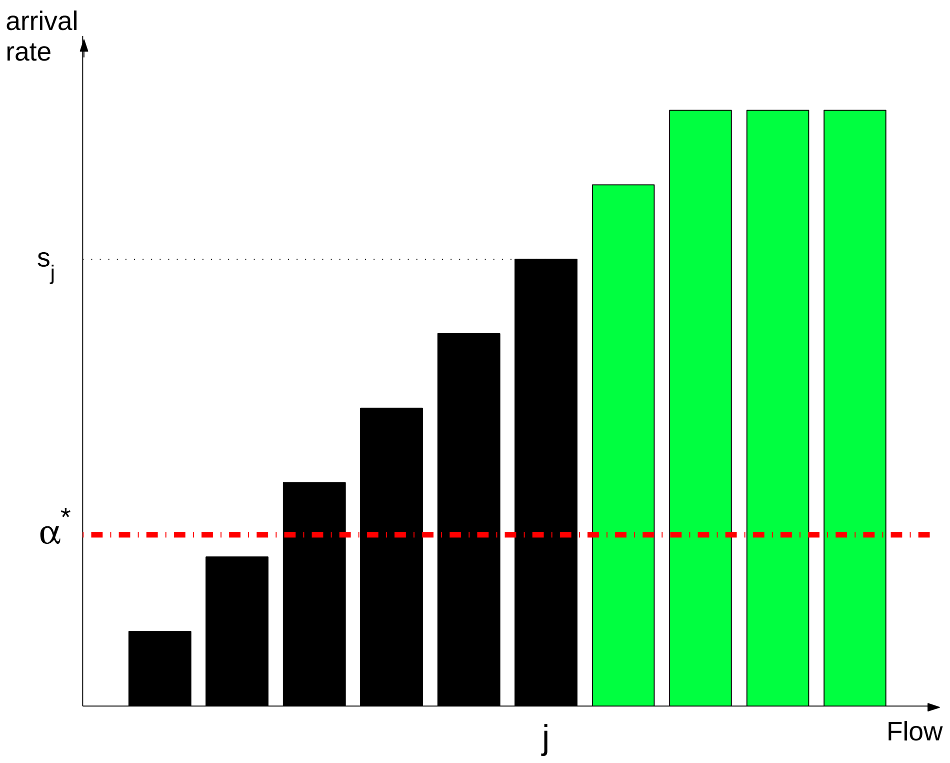


Figure 6.1: The relationship between the “max-min-fairness” threshold α^* and the ideal value s_j of our estimate α. Each bar represents the arrival rate of a flow. The total area in black is at most C; and if we change other bar, not in black, into black, then the total area in black is greater than C. Our estimate α will fluctuate about s_j, that is greater than α^*.

in the new *good region* — we call this time the **settling time**. We will show that this *settling time* is acceptably short. In other words, the system will stabilize quickly. There are two cases here:

1. If the current estimated value of α is greater than or equal to the new "max-min-fairness" threshold α^*_{new}, then the system is still in the *good region.* Thus, the *settling time* is 0.

2. If the current estimated value of α is less than the new "max-min-fairness" threshold α^*_{new}, (obviously, α^*_{new} is greater than the original one α^*_{old},) there are two subcases: (when α^* is changed)

   (a) If there is no flow whose arrival rate is greater than α but at most α^*_{new}, then there is no flow punished wrongly, and the system is still in the *good region.*

   (b) If there is at least one flow whose arrival rate is in (α, α^*_{new}], the worst case is that when α^* is changed, $q = q_{max} = E + \log C \cdot (S + C) \cdot \Delta t$. So the *settling time* includes two parts: the time for q becoming less than E, and the time for α incrementing to be greater than α^*_{new}.

   The time for q becoming less than E is at most $\frac{\log C \cdot (S+C) \cdot \Delta t}{\alpha} \leq \frac{\log C \cdot (S+C) \cdot \Delta t}{\alpha^*_{old}}$.

   The system will start incrementing α when q becomes less than E. The incrementing time until α is greater than α^*_{new} is $\lceil \log_2 \frac{\alpha^*_{new}}{\alpha_0} \rceil$, which is at most $\lceil \log_2 \frac{\alpha^*_{new}}{\alpha^*_{old}} \rceil$. (Let α_0 be the threshold value when q becomes less than E.)

   Therefore, the total *settling time* is at most

$$\frac{\log C \cdot (S + C) \cdot \Delta t}{\alpha^*_{old}} + \lceil \log_2 \frac{\alpha^*_{new}}{\alpha^*_{old}} \rceil.$$

Therefore, we have the following theorem:

Theorem 16 *When the arrival rates of flows are changed, the settling time of pro-*

tocol FBA is at most

$$\frac{\log C \cdot (S + C) \cdot \Delta t}{\alpha^*_{old}} + \lceil \log_2 \frac{\alpha^*_{new}}{\alpha^*_{old}} \rceil.$$

Chapter 7

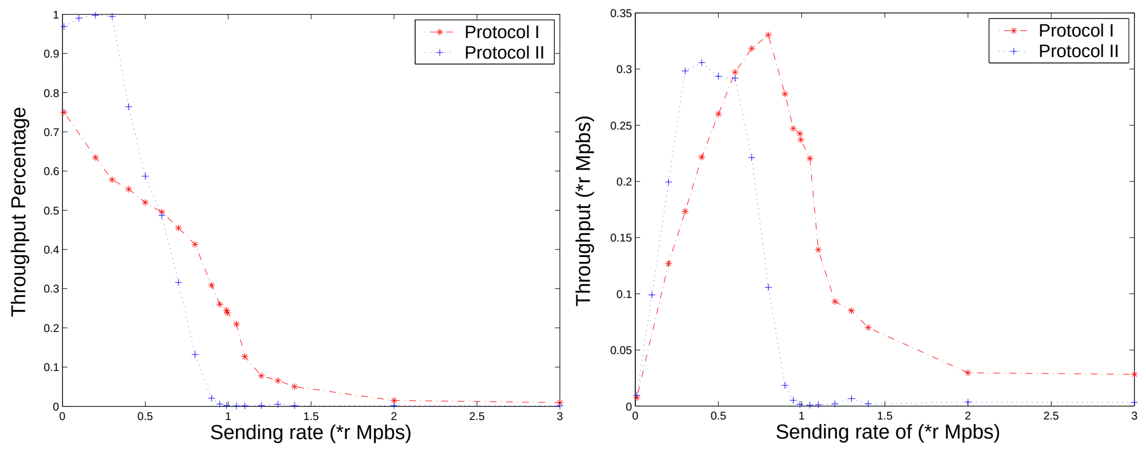
Simulations and relative advantage of protocols I, II, and FBA

We evaluate the performance of our protocol by simulations under various network configurations and parameters. Overall, protocols I , II, and FBA achieve reasonably fair bandwidth allocations. Protocols II and FBA do 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, but it still must be addressed since we cannot prevent sources from not optimizing their utilities).

7.1 Testing the equilibrium property for protocols I and II

To verify the Nash equilibrium achieved by the protocol, we consider a single r Mbps congested link shared by 5 Poisson flows, indexed from 0 to 4. The sending rates of flow 0, 1, 2, and 3 are fixed: r Mbps, *2r/3* Mbps, *r/2* Mbps, and *2r/5* Mbps, while flow 4 is allowed to send packet at arbitrary rates. We vary the sending rate of flow 4 to study the performance of our protocol over a *5000r* sec interval.

When the buffer size is set to be infinite, $H = F/6$, and L is 0, the simulation results for protocols I and II are summarized in Figure 7.1. As the sending rate of flow 4 increases, the dropping percentage goes up as well. When the sending rate reaches the maximal value, r, a large percentage of the packets are dropped; but below that rate, a large percentage of the packets go through. The greatest bandwidth, approximately *0.3r* Mbps, is obtained at a sending rate less than r Mbps.



(a) (b)

Figure 7.1: Simulation results for a finite buffer under protocols I and II. (a) Throughput percentage of packets from flow 4, as a function of the sending rate. (b) Average throughput of flow 4, as a function of the sending rate.

In the following simulations we use the NS simulator, available at [1], to test more complicated behaviors. For TCP flows, we will use NewReno TCP protocol [10].

7.2 Testing the queue length property of protocol FBA

Before giving the simulation result, let us first analyze this special case.

Suppose there are n flows. All flows are arriving at the same rate s. When the total arrival traffic is less than the capacity C, there is no congestion and we don't need to use the threshold α controller to control the congestion.

When the total arrival traffic is greater than the capacity C, congestion happens and since all flows are arriving at the same rate, we may write the operations of control system as the following steps: (it starts with threshold $\alpha = C$ and the number of *SEND* packets in the queue $q = 0$)

1. α remains unchanged till $q = E$;

2. Update α using formula $\alpha := \frac{C}{C+q'}\ \alpha$ till $q' \leq 0$; (In other words, updating α using formula $\alpha := \frac{C}{n \cdot s}\ \alpha$ till $\alpha \leq s$)

3. α keeps unchanged till $q = E$;

4. update α using formula $\alpha := 2\alpha$ till $q' \geq 0$ (or $\alpha > s$);

5. goto 1.

We assume that each packet has 8000 bits. Let the router capacity C be 150 packets/sec (1.2 Mbps), the arrival rate s of each flow be 10 packets/sec (0.08 Mbps), and the number of flows be $n = 20$. We update α every $\Delta t = 1/C$ sec. From the configuration,

1. $S \geq n \cdot s$ =200 packets/sec; let $S = 250$ packets/sec = 2 Mbps.

2. $E \geq 2 \log 2C \cdot C \cdot \Delta t = 11.4076$ packets; let $E = 15$ packets.

3. $F \geq (E + \log C \cdot (S + C) \cdot \Delta t) \cdot (2S - C)/C = 79.9793$ packets.

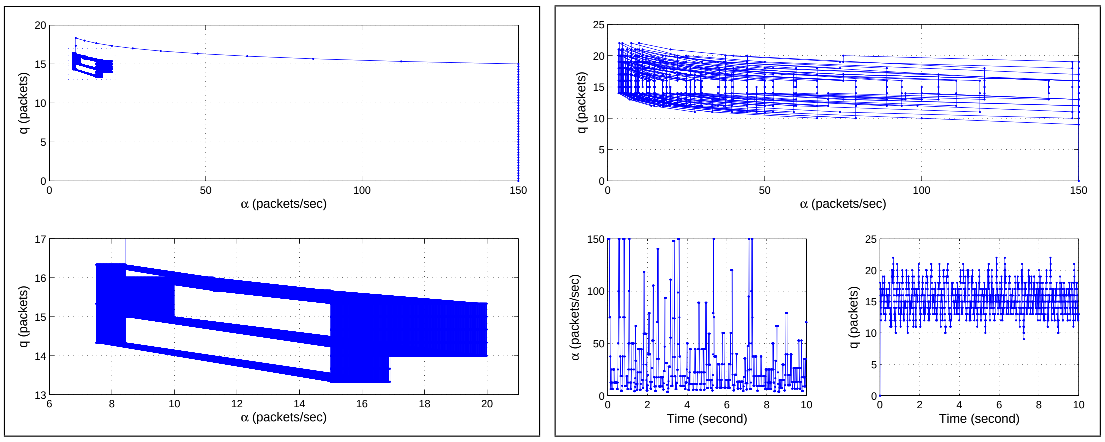
Using theoretical computation, we can plot the predicted behavior of the system. The theoretical result is shown in Figure 7.2(a) with the plots of points (α,q).

By simulation, over a 10 second interval, the set of points (α,q) is plotted in the upper figure of Figure 7.2(b). The lower two figures of Figure 7.2(b) show the values of α and q as a function of time. The average throughput of each flow is approximately 7.5 packets/sec[1].

It turns out that when we update according to the formula $\alpha := 2\alpha$, α increases too fast and the queue length reaches capacity frequently. The situation is improved with slower growth rate: we include simulation results using parameters 1.5 and 1.2 in Figure 7.3.

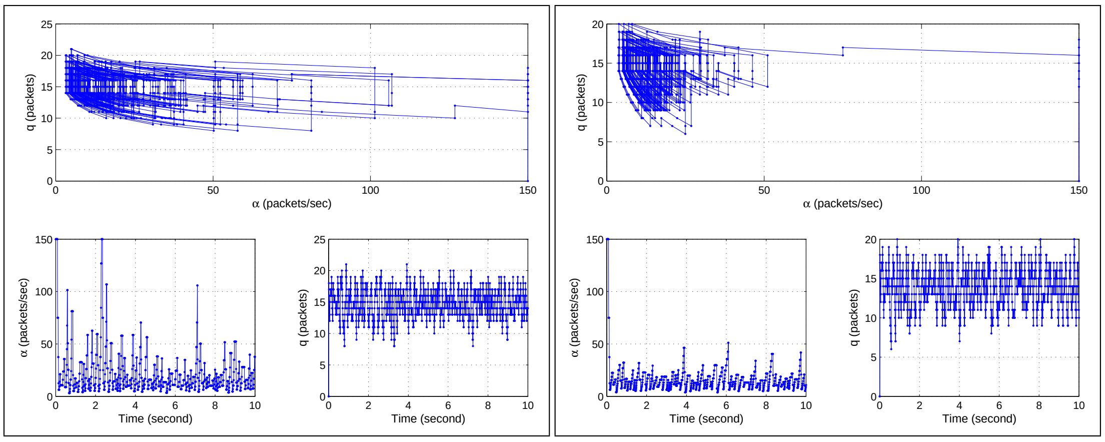
Although the results of theoretical computation and simulation are not the same, they are close. The reason for the difference between theory and simulation is that, in theory we are assuming that we know the exact value of each source rate, but in simulation the value we get for a variable is just an approximation.

[1]The exact values of the throughputs are 7.4, 8.3, 7.6, 7.1, 7.1, 7.4, 7.7, 7.9, 7.7, 7.5, 7.7, 7.6, 7.8, 7.3, 7.4, 7.8, 6.9, 6.9, 7.4, and 7.6 packets/sec.



(a) (b)

Figure 7.2: All flows are arriving at the same rate: (a) Theoretical computation result of points (α,q). The lower graph is the magnification of the upper graph when $6 \leq \alpha \leq 21$ and $13 \leq q \leq 17$. (b) Simulation result of points (α,q) is plotted in the upper figure; the lower two figures show the values of α and q as a function of time.



(a) (b)

Figure 7.3: All flows are arriving at the same rate: (a) Simulation result when we use $\alpha := 1.5\alpha$ to increase α. (b) Simulation result when we use $\alpha := 1.2\alpha$ to increase α.

7.3 Performance on a single congested link

7.3.1 Comparison with CHOKe

To compare the performances of our protocols and CHOKe on a single congested link, we consider a 150 packets/sec (1.2 Mbps[2]) link shared by one UDP flow (indexed 1) which is arriving at 150 packets/sec (1.2 Mbps), and 32 TCP flows (indexed from 2 to 33) whose propagation delays are 3 ms. For protocol FBA, we set E to be 15 packets and update α every $\Delta t = 1/C$ sec.

The average throughput of each flow over a 100 sec interval is given in Figure 7.4. We notice that the ill-behaved UDP flow is penalized heavily (its throughput is much less than its arrival rate) and most of the TCP flows get approximately their fair rates under protocols I, II and FBA; but CHOKe doesn't provide as much bandwidth to each TCP source as to the UDP source.

7.3.2 Ten UDPs and ten TCPs flows on a single congested link

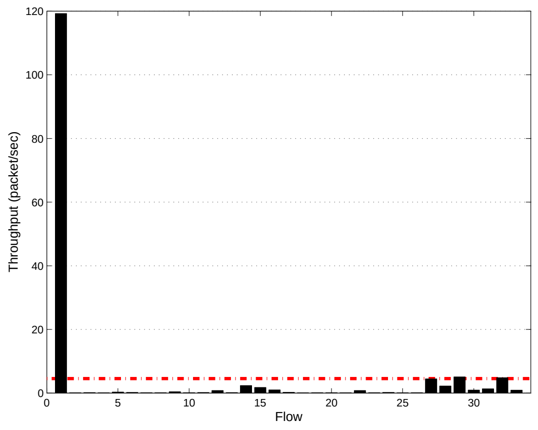
To examine the impact of a set of ill-behaved UDP flows on a set of TCP flows, we consider a 150 packets/sec (1.2 Mbps) link shared by ten UDP sources[3] and ten TCP sources whose propagation delays are set to be 3 ms. For protocol FBA, we still set E =15 packets and update α every $\Delta t = 1/C$ second.

We compare the performances of CHOKe, protocols I, II, FBA, and FBA with parameters 1.5 (using $\alpha := 1.5\alpha$ to increase α) and 1.2 (using $\alpha := 1.2\alpha$ to increase α). The average throughput of each flow over 100 seconds is shown in Figure 7.5. And the maximal and minimal throughputs of two kinds of flows over a 100 sec interval is given in Table 7.1[4]. (Simulations of traditional TCP, Reno, and Sack gave similar

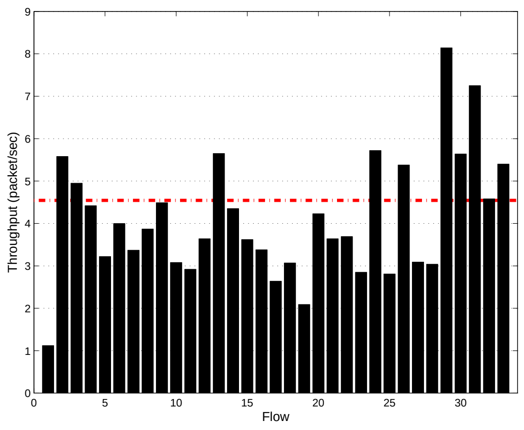
[2]A packet has 8000 bits.

[3]Their arrival rates are 150, 125, 100, 87.5, 75, 62.5, 50, 37.5, 25, and 12.5 packets/sec respectively.

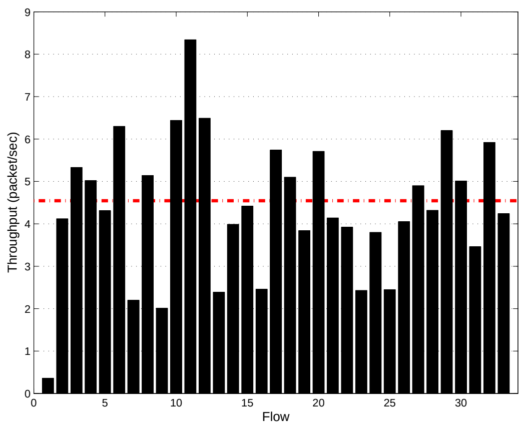
[4]We may notice that the minimal throughput of TCP flows under FBA is much less than its maximal throughput. But for other versions of TCP, such as Reno and Sack, there is no such big difference. For example, with TCP Sack, the minimal throughput is 2.76 packets/sec and the maximal throughput is 6.05 packets/sec.



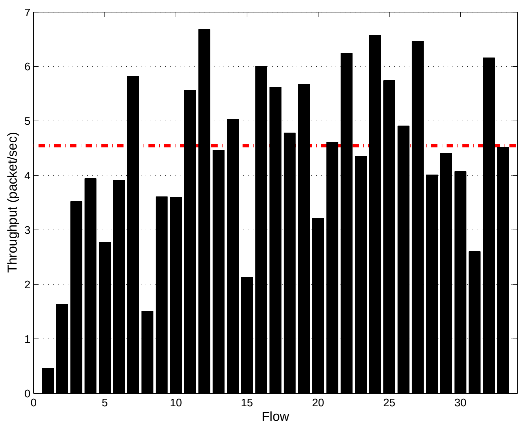
(a) CHOKe



(b) Protocol I



(c) Protocol II



(d) Protocol FBA

Figure 7.4: Simulation result of one UDP (flow 1) and 32 TCP flows on a single congested link. The average throughput of each flow over a 100 sec interval is plotted under (a) CHOKe, (b) protocol I, (c) protocol II, and (d) protocol FBA. The first bar represents the throughput of the UDP flow, the other bars represent throughputs of TCP flows, and the dash-dot line shows their fair-share rate.

Protocol	CHOKe	I	II	FBA	FBA(1.5)	FBA(1.2)
UDP MAX (packets/sec)	34.36	31.75	21.04	13.00	15.30	16.79
UDP MIN (packets/sec)	2.3	1.39	2.63	1.78	0.17	0.12
TCP MAX (packets/sec)	0.03	0.16	0.64	6.07	6.90	8.52
TCP MIN (packets/sec)	0.01	0.08	0.08	0.41	3.10	7.07

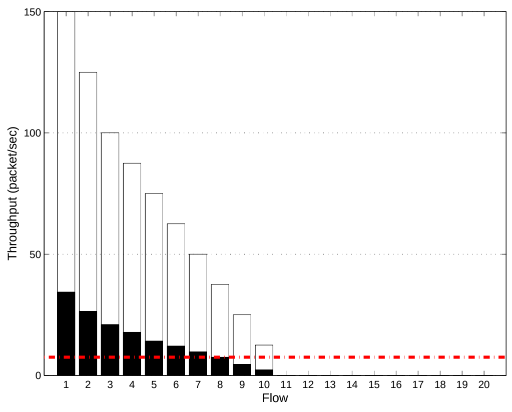
Table 7.1: Simulation result of ten UDP and ten TCP flows on a single congested link. The maximal and minimal throughputs of two kinds of flows over a 100 sec interval under CHOKe, protocols I, II, FBA, and FBA with parameters 1.5 and 1.2.

results.) 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 CHOKe, protocols I and II don't penalize the ill-behaved UDP flows enough as they still get more than their fair rates, and protocol FBA stands out by providing more bandwidth to the TCP sources than other protocols. The result under FBA is further improved with slower growth rate (1.2).

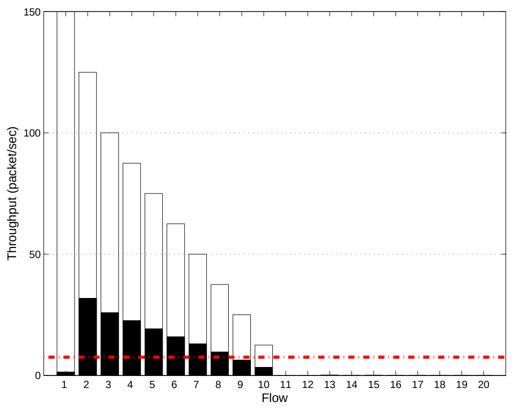
7.3.3 Impact of four ill-behaved UDP flows on a single congested link

In this section, we examine the impact of four ill-behaved UDP flows on a single TCP flows on a 150 packets/sec (1.2 Mbps) link. Each UDP flow has an arrival rate of 100 packets/sec (1.2 Mbps). The propagation delay of TCP flow is 3 ms. For protocol FBA, we still set E =15 packets and update α every $\Delta t = 1/C$ second.

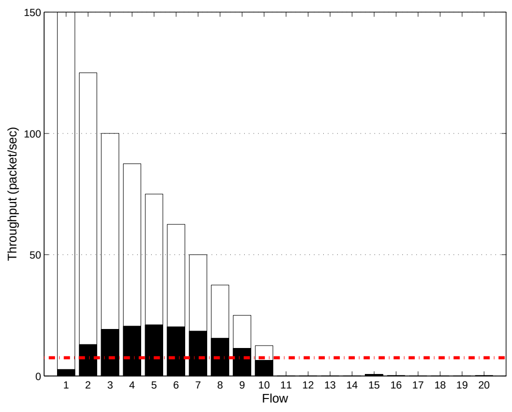
We compare the performances of CHOKe, protocols I, II, and FBA. The average throughput of each flow over 100 seconds is shown in Figure 7.6, which is also listed in Table 7.2. All of the protocols prevent the TCP sources from being entirely shut out of the channel (as would happen without any penalties to unresponsive flows); but CHOKe and protocol I don't penalize the ill-behaved UDP flows enough as they still get more than their fair rates. Although under protocols II and FBA, TCP flow still gets less than its "max-min-fairness" rate, it achieves a reasonable degree of fairness.



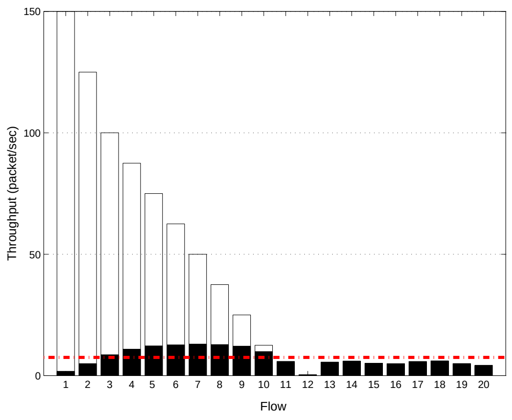
(a) CHOKe



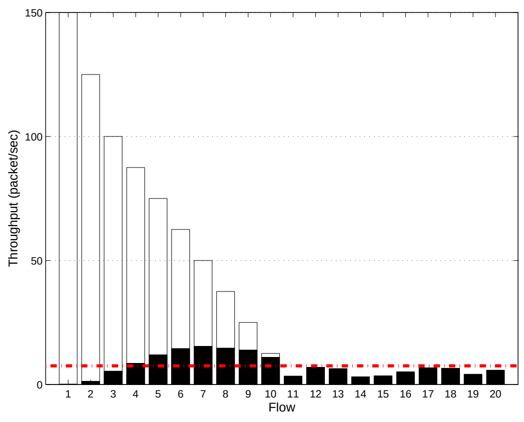
(b) Protocol I



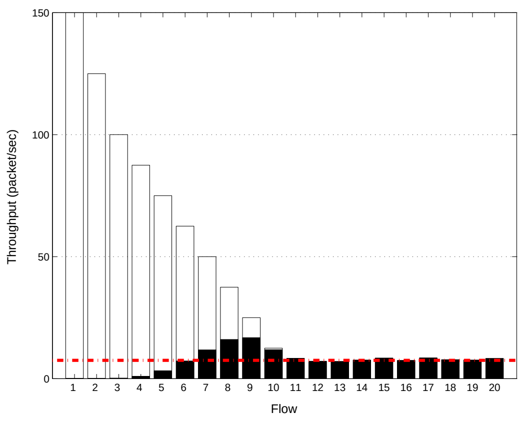
(c) Protocol II



(d) Protocol FBA



(e) Protocol FBA with parameter 1.5



(f) Protocol FBA with parameter 1.2

Figure 7.5: Simulation result of ten UDP and ten TCP flows on a single congested link. The average throughput of each flow over a 100 sec interval is plotted under (a) CHOKe, (b) protocol I, (c) protocol II, (d) protocol FBA, (e) protocol FBA using $\alpha := 1.5\alpha$ to increase α, and (f) protocol FBA using $\alpha := 1.2\alpha$ to increase α. Each black bar represents the throughput of a flow: the first ten bars (from 1 to 10) are for UDP flows and the other ten (from 11 to 20) are for TCP flows. The bars without color represent the arrival rates of UDP flows. The dash-dot line shows the fair-share rate.

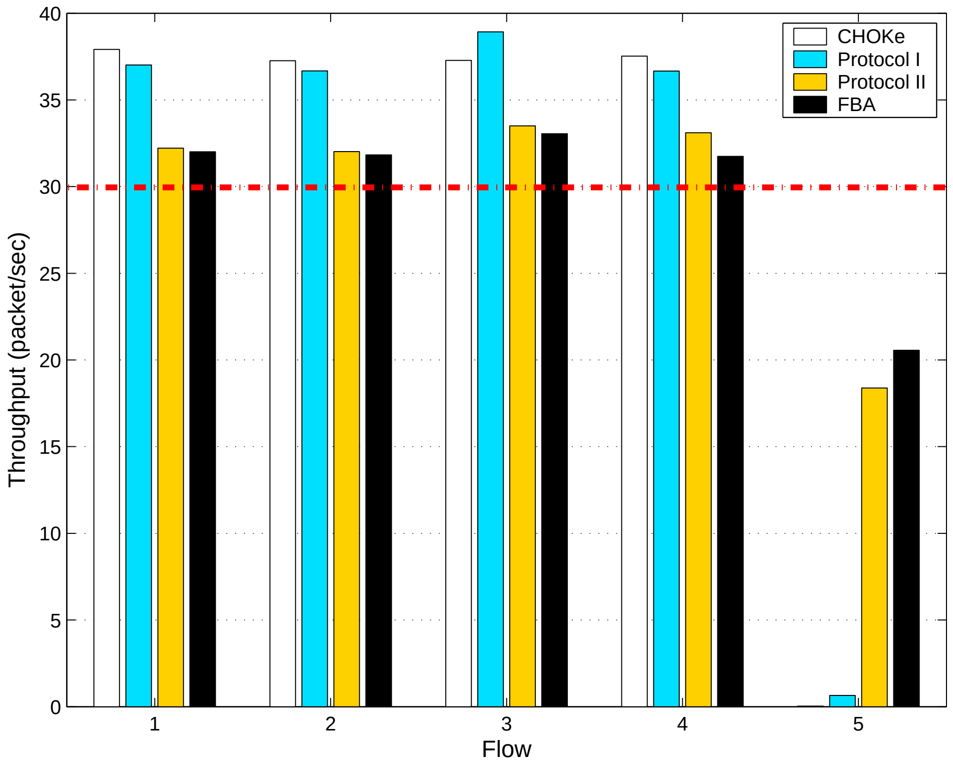


Figure 7.6: Simulation result of four UDP flows and one TCP flow on a single congested link of 150 packets/sec (1.2 Mbps). The average throughput of each flow over a 100 sec interval is plotted. Each bar represents the throughput of a flow: the first four bars (from 1 to 4) are for UDP flows and the last one (from 11 to 20) is for the TCP flow. (For each flow, there are four bars there. From left to right, they are under CHOKe, protocols I, II, and FBA.)

Protocol	CHOKe	I	II	FBA
UDP0 (packets/sec)	37.91	37.01	32.22	32.01
UDP1 (packets/sec)	37.26	36.67	32.02	31.83
UDP2 (packets/sec)	37.28	38.93	33.50	33.05
UDP3 (packets/sec)	37.53	36.66	33.11	31.74
TCP4 (packets/sec)	0.03	0.650	18.38	20.56

Table 7.2: The average throughputs of four UDP flows and one TCP flow, on a 150 packets/sec (1.2 Mbps) link, over a 100 sec interval under CHOKe, protocols I, II, and FBA.

7.4 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 7.7. 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). There are three flows in the network. In addition, for protocol FBA, we set E =25 packets and update α every $\Delta t = 1/C$ second. Based on the network configuration, we perform the following experiments.

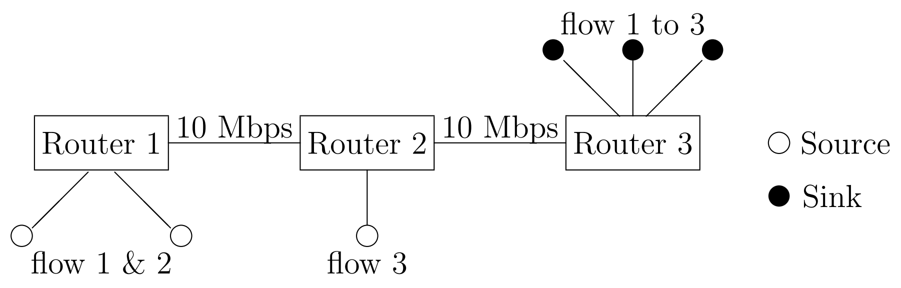


Figure 7.7: Topology for analyzing how the throughput of a flow is affected by more than one congested link.

7.4.1 Three UDP flows

When three flows are all using UDP protocol and send at 10 Mbps, links L12 and L23 will both become congested. Over a 20 sec interval, the average throughputs of flow 1 and 2 on link L12 are listed in Table 7.3; the average throughputs of flow 1, 2, and 3 on link L23 are listed in Table 7.4. On link L12, the “max-min-fairness” rates of flow 1 and 2 are both 5 Mbps; since they both send at 10 Mbps, the throughputs they get are close to 5 Mbps. On link L23, flow 3 sends at 10 Mbps, which is much higher than the rate of other two flows, so flow 3 is penalized and gets less throughput under protocols I and II.

	CHOKe	Protocol I	Protocol II	FBA
UDP 1 (Mbps)	5.089	4.799	4.819	4.938
UDP 2 (Mbps)	4.910	5.201	5.181	4.963

Table 7.3: Three UDP flows. The average throughputs of flow 1 and 2 on link L12 is given under CHOKe, protocols I, II, and FBA.

	CHOKe	Protocol I	Protocol II	FBA
UDP 1 (Mbps)	2.262	4.701	4.721	3.394
UDP 2 (Mbps)	2.196	5.098	5.082	3.334
UDP 3 (Mbps)	5.539	0.199	0.195	3.270

Table 7.4: Three UDP flows. The average throughputs of flow 1, 2, and 3 on link L23 is given under CHOKe, protocols I, II, and FBA.

7.4.2 Three TCP flows

Next, three flows are all using TCP protocol and the propagation delays of all TCPs are 1 ms. Over a 20 sec interval, the average throughputs of flow 1 and 2 on link L12 are listed in Table 7.5; the average throughputs of flow 1, 2, and 3 on link L23 are listed in Table 7.6. Since TCP flows are responsive to congestion and adjust their sending rates accordingly, it is reasonable that each TCP flow losses a quite small number of packets and gets close to its fair rate.

	CHOKe	Protocol I	Protocol II	FBA
TCP 1 (Mbps)	2.492	3.087	3.216	2.444
TCP 2 (Mbps)	3.928	3.969	3.105	2.445

Table 7.5: Three TCP flows. The average throughputs of flow 1 and 2 on link L12 is given under CHOKe, protocols I, II, and FBA.

	CHOKe	Protocol I	Protocol II	FBA
TCP 1 (Mbps)	2.443	2.979	3.031	2.443
TCP 2 (Mbps)	3.871	3.795	2.934	2.445
TCP 3 (Mbps)	2.824	2.366	3.180	2.446

Table 7.6: Three TCP flows. The average throughputs of flow 1, 2, and 3 on link L23 is given under CHOKe, protocols I, II, and FBA.

7.4.3 One UDP flow and two TCP flows

Now, flow 1 is a UDP source, which sends at 10 Mbps, and the other two are TCP flows. The propagation delays of TCPs are 1 ms. Over a 20 sec interval, the average throughputs of flow 1 and 2 on link L12 are listed in Table 7.7; the average throughputs of flow 1, 2, and 3 on link L23 are listed in Table 7.8. Since there are two links, the throughput of TCP 2 is dominated by link L23. The performance of protocol I and protocol II are comparable and they outperform CHOKe and protocol FBA. Although protocol FBA is not comparable to protocols I and II, it is still better than CHOKe.

	CHOKe	Protocol I	Protocol II	FBA
UDP 1 (Mbps)	9.491	7.112	7.976	8.596
TCP 2 (Mbps)	0.422	2.763	1.927	0.626

Table 7.7: One UDP flow and two TCP flows. The average throughputs of flow 1 and 2 on link L12 is given under CHOKe, protocols I, II, and FBA.

	CHOKe	Protocol I	Protocol II	FBA
UDP 1 (Mbps)	9.033	4.540	4.620	7.247
TCP 2 (Mbps)	0.406	2.685	1.856	0.625
TCP 3 (Mbps)	0.481	2.563	3.316	2.021

Table 7.8: One UDP flow and two TCP flows. The average throughputs of flow 1, 2, and 3 on link L23 is given under CHOKe, protocols I, II, and FBA.

Chapter 8

Conclusions and future work

We have argued in this thesis on the importance of mechanisms in the network to control congestion. We propose router congestion control schemes, which aim to approximate “max-min-fairness” with low implementation overhead. Some analytical results were derived for their fairness, efficiency, and stability. Simulation results suggest that our schemes work well in protecting congestion-sensitive flows from congestion-insensitive flows and achieve a higher degree of fairness than the previous best protocol CHOKe.

There is more work to be done, such as studying the performance of the algorithm under a wider range of network topologies and real traffic traces, evolving the details of the design, and considering hardware implementation issues. Another matter which has not been dealt with in our (or any other) work is the possibility of collusion among sources.

However, the main point of our thesis is the overall idea of using game theory and auction theory in router congestion control. The detailed algorithm proposed here represents only initial prototypes. We expect that the thesis will provide an incentive for future work on this problem. It’s a long way to go.

Bibliography

[1] Ns (network simulator), 1995. URL: http://www.isi.edu/nsnam/ns/.

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

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

[4] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall, 1992.

[5] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. *IETF RFC (Informational) 2309*, April 1998.

[6] B. Braden and Ed. Requirements for internet hosts – communication layers. *STD 3, RFC 1122*, October 1989.

[7] 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 3-12.

[8] S. Floyd and K. Fall. Router mechanisms to support end-to-end congestion control. *LBL Technical Report*, February 1997.

[9] 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.

[10] S. Floyd, T. Henderson, and A. Gurtov. The newreno modification to tcp's fast recovery algorithm. *RFC 3782*, April 2004.

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

[12] S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proc. ACM SIGCOMM*, pages 342–355, 1996.

[13] 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.

[14] X. Gao, K. Jain, and L. J. Schulman. Fair and efficient router congestion control. In *15'th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.

[15] S. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proc. IEEE INFOCOM*, pages 636–646, 1994.

[16] E. L. Hahne. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal of Selected Areas in Communications*, 9(7):1024–1039, 1991.

[17] E. L. Hahne and R. Gallager. Round-robin scheduling for fair flow control in data communication networks. *Report LIDS-P-1537.*

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

[19] V. Jacobson. Congestion avoidance and control. In *SIGCOMM '88 Symposium on Communications Architectures and Protocols*, August 1988. In *Computer Communication Review,* vol. 18 no. 4, 314-329, August 1988.

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

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

[22] S. Keshav. A control-theoretic approach to flow control. In *Proc. ACM SIGCOMM*, pages 3–15, September 1991.

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

[24] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *9th International Conference on Network Protocols (ICNP)*, November 2001.

[25] A. Manin and K. Ramakrishnan. Gateway congestion control survey. *IETF RFC (Informational) 1254*, August 1991.

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

[27] J. Nagle. On packet switches with infinite storage. *IEEE Trans. on Comm.*, 35(4), Apr 1987.

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

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

[30] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Trans. Netw.*, 1(3):344–357, 1993.

[31] B. Radunovic and J. Le Boudec. A unified framework for max-min and min-max fairness with applications, 2002.

[32] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proc. ACM SIGCOMM*, pages 231–242, 1995.

[33] 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.

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

[35] J. Wang, A. Tang, and S. H. Low. Maximum and asymptotic udp throughput under CHOKe. In *Proc. ACM Sigmetrics*, 2003.

[36] L. Zhang. Virtual clock: a new traffic control algorithm for packet switching networks. In *Proceedings of the ACM symposium on Communications architectures & protocols*, pages 19–29. ACM Press, 1990.