Adaptive Modification of Generalized Processor Sharing*

Richard J. La, and Venkat Anantharam
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
hyongla@eecs.berkeley.edu, ananth@eecs.berkeley.edu

November 9, 1998

Abstract

Communication networking has been one of the most active research areas in the past few years. With the emergence of many real time applications, the need for faster networks is becoming increasingly apparent. As a result, there has been a lot of research activity aimed at finding better multiplexing schemes that can handle the traffic from a wide range of applications from real-time applications such as video and audio to e-mail that is much less sensitive to delays. Along with some other work conserving service disciplines, Generalized Processor Sharing (GPS) has been introduced as a multiplexing scheme that can provide worst case delay guarantees, and appears to be quite popular.

In this paper we investigate the possibility of improving the Packet-by-packet Generalized Processor Sharing (PGPS) scheme, by using an idea from zero-sum game theory to find strategies that adapt to the history of the sources by maintaining a state called the virtual backlog. Due to the complexity involved in the computation of the optimal strategies, most of the work done here is for the case with two input sessions. We mainly focus on the analysis of a fluid model and carry out simulations of PGPS and the proposed modification, using the network traffic data samples taken at Bellcore.

1 Introduction

With the exponential growth of the Internet, network designers are faced with many interesting problems as the demand for bandwidth increases rapidly with no signs of slowing down. One of the problems network designers are faced with in order to meet the demand is finding a multiplexing scheme that can handle different types of applications efficiently. This is because consumers demand that the network

*Research support by NSF grant NCR 9422513.

be able to handle a wide range of different applications, from real time video conferencing to e-mail, which is much less sensitive to delay.

Differing demands on the network can be handled by offering services with different performance guarantees, i.e., with different quality of service (QoS). There is general agreement that as a part of the QoS, the network should be able to provide an upperbound on the maximum packet delay in most cases. This poses a challenging problem for network designers, because in the attempt to provide a strict upperbound on the delay, it may not be possible to achieve the high statistical multiplexing gain that results in efficient use of the network resources.

Many new multiplexing schemes have been proposed in the past to address this issue. Multiplexing schemes can be categorized into two groups: work conserving and non-work conserving. Work conserving service disciplines never stay idle when there are packets to be transmitted, while non-work conserving service disciplines sometimes stay idle in order to reduce the delay jitter. Here we will briefly survey some of the most important work conserving service disciplines that have been studied so far.

Zhang [16] introduced a scheme called *Virtual Clock Multiplexing*. Virtual clock multiplexing provides a guaranteed rate and average delay for each session. The virtual clock discipline aims at emulating the time division multiplexing (TDM) scheme. Each packet is assigned a virtual transmission time, which is the time at which the packet would have been transmitted were the server implementing TDM. Packets are transmitted in the increasing order of virtual transmission times. One of the problems with this scheme is that if a session generates a big burst of data at some point even when the system is lightly loaded, the session can be punished much later in time when other sessions become more active.

There is another scheme called *Weighted Round Robin* [9]. Under weighted round robin, every session i is given an integer weight w_i associated with it, and the server polls the sessions according to a predetermined sequence, trying to serve session i at a rate of $\frac{w_i}{\sum_j w_j}$. If an empty buffer is encountered, the server instantaneously moves to the next session in the order. When an arriving session i packet just misses its slot in a frame, it cannot be transmitted before the next session i slot. If the system is heavily loaded in the sense that almost every slot is utilized, the packet may have to wait almost N slot times to be served, where N is the number of sessions sharing the server.

Another interesting service discipline is *Delay-Earliest-Due-Date* or delay-EDD, which is an extension of the classic Earliest-Due-Date-First (EDD) scheduling [8]. In the original EDD, each packet from a periodic traffic stream is assigned a deadline and the packets are sent in the order of increasing deadlines. In delay-EDD, the server negotiates a service contract with each source, and if a source obeys the traffic specifications under the contract, such as a peak and average rate, then the server

provides a delay bound. The key lies in the assignment of deadlines to packets. The server sets the deadline of a packet to the time at which it should be transmitted had it been received according to the contract, which is the sum of the expected arrival time and the delay bound at the server.

We now turn to Generalized Processor Sharing (GPS) or Fluid Fair Queueing (FFQ), which appears to be the most popular among the multiplexing schemes that have been suggested in the past. GPS is a general form of the head-of-line processor sharing service discipline (HOL-PS) [10]. With HOL-PS, there is a separate FIFO buffer for each session sharing the same link. During any time interval when there are exactly N nonempty queues, the server serves the N packets at the head of the queues simultaneously, each at a rate of one Nth of the service rate. While a HOL-PS server serves all nonempty queues at the same rate, GPS allows different sessions to have different service shares. A GPS server is characterized by N positive real numbers, $\phi_i, i = 1, \ldots, N$. Under GPS the delay of session i packet can have an upperbound given in terms of session i's queue size at the time of arrival without any rate control mechanism. This feature enables the sessions to take advantage of lightly loaded network conditions. Also, since GPS is work conserving, it is an efficient multiplexing scheme. Parekh and Gallager [12, 13] have done an extensive analysis of GPS. They have shown that there exist relatively tight upper bounds on the session i packet delay and backlog in terms of ϕ_i's, arrival rates, and the service rate.

In this paper, we attempt to improve on GPS using some ideas from game theory. Rather than attempting to analyze an entire network, we will focus on one switch, i.e., a single node. In section 2, we will show that the scheduling problem at the switch can be modeled as a zero-sum stochastic game with a cost function defined as a function of a certain state, certain actions taken by the controllers, and the incoming flow. There are two players in this game. Player 1 is the controller, who attempts to minimize the payoff to player 2, who represents the users, who in turn want to maximize the payoff from the controller. The optimal strategy for the controller as well as the optimal strategy for the users can be calculated by using the Shapley recursion [14], which will be discussed in section 5.

Throughout the paper, we will assume that the traffic admitted into the network is shaped by the leaky bucket flow control mechanism before it is admitted. The key idea of this paper is that if the traffic offered to the network is shaped by the leaky bucket scheme, it is possible to summarize the past of the traffic with a simple recursively updatable statistic called the *virtual backlog*. This term was first introduced by Anantharam [2] where the potential value of this idea in adopting a game-theoretic approach to resource scheduling problems in networks was also pointed out.

We test the new multiplexing scheme we come up with in this paper using simulation. For the purpose of the simulation, we limited ourselves to the case with two input sessions, and the optimal strategy for the controller was computed off-line by using Shapley recursion method and implemented

in the simulation with Discrete-time Packet-by-packet GPS (DTPGPS) with the necessary modifications. DTPGPS and the modified DTPGPS are described in section 6.

The paper is organized in the following way. First we will describe the model and leaky bucket flow control scheme. After that, there will be a brief discussion on stochastic games and Shapley recursion. In section 5 we will explain DTPGPS and its modification, and then describe the computation of the optimal strategies. We will discuss and analyze a fluid model for our multiplexing scheme in section 7. Then we will present the simulation results, followed by the conclusion.

2 Model

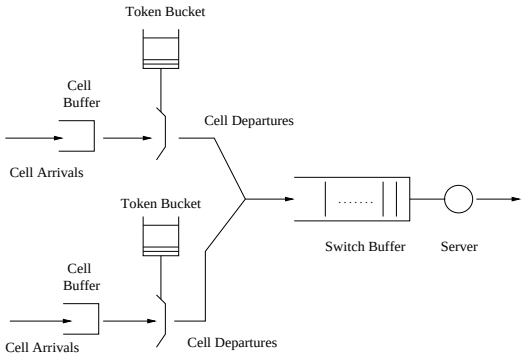

Figure 1: A switch with two input sessions.

In this section we describe the model and formulate the problem. Due to the complexity of optimal strategies, as will be explained in the following sections, instead of looking at the entire network, we will only look at the simple case of one switch with two sources, as shown in Figure 1. We will be using a discrete time model throughout the paper.

We assume that the leaky bucket scheme is used to control the burstiness of the arriving traffic before packets are admitted into the network. The arriving packets from each input session get stored in the buffer for that session. If there is a token available in the token bucket when a packet arrives, then it is immediately admitted into the network, consuming one token. Otherwise, it has to wait till a token is available. Once a packet is admitted to the network, it is stored in the switch buffer for the session and waits till it is scheduled to be served by the server. The server has a fixed service rate, r Mbps. Both the token buckets and the switch buffers are finite.

It is proved in [1] that the entire history of a burstiness constrained flow can be kept track of by the *recursively updatable* virtual backlog, which is a simple finite dimensional statistic. We investigate the possibility of improving the efficiency of the switch by designing a switch that dynamically adapts to the current state, using the virtual backlog.

Since we are using a discrete time model, we assume that the *state* of the network element at time $n, n = 0, 1, \cdots$, is given by an element $\xi_n \in \Xi$, where Ξ is a set called the state space of the network element. This set includes occupancies of the switch buffers as well as any memory in the scheduling algorithm. At time n the network element chooses a *control action* $u_n \in U$, where U is the set of feasible actions. A *message flow* is a sequence of nonnegative real numbers $(a_n, n \geq 0)$. We model traffic on the links of a network by a message flow. The evolution of the state of the network element occurs in response to the incoming flows $\underline{a}_n$ at the current time n and the control action chosen at each time n yielding the evolution equation

$$\xi_{n+1} = f(\xi_n, u_n, \underline{a}_n) \tag{1}$$

A message flow $(a_n, n \geq 0)$ is said to be (σ, ρ) constrained if for all $0 \leq n_0 \leq n_1 < \infty$, we have

$$\sum_{k=n_0}^{n_1} a_k \leq \sigma + \rho(n_1 - n_0 + 1). \tag{2}$$

We assume that the traffic flow from ith source is (σ^i, ρ^i) constrained, $1 \leq i \leq 2$. The domain of f is $\Xi \times U \times \prod_{i=1}^{2}[0, \sigma_i + \rho_i]$.

The problem of designing a good adaptive controller strategy is modeled as a zero-sum stochastic game with two players. A cost function for the controller is defined as a function of action, state, and message flows. There are two parts to the cost function. First, there is a cost for holding packets in the input buffer, and second, there is a cost for dropping packets when the buffers are full. Since dropping a packet causes a relatively long effect, the cost for dropping a packet is relatively big compared to the cost for holding packets in the buffers. This suggests that the switch should attempt to minimize the number of packets that get dropped.

Player 1, which is the controller, tries to minimize the amount it has to pay to the users, and player 2, that represents the users, tries to maximize the payoff from the controller. We then formulate the problem of the controller as one of minimizing the total infinite horizon discounted cost. Since we have a convex action space, this is equivalent to a worst case formulation where the controller attempts to minimize the overall discounted cost over all possible burstiness constrained source sequences.

It is shown in [14] that the stochastic game above admits a continuous value function that is the unique fixed point of an associated Shapley recursion. Moreover, both the controller and the users have optimal stationary randomized adaptive strategies which depend only on the state of the game.

In our model the state, ξ, consists of four parameters; x_1 is the number of session 1 backlogged packets in the switch buffer, x_2 is the number of session 2 backlogged packets in the switch buffer, θ_1 is the number of tokens available for session 1 in the leaky bucket, and θ_2 is the number of tokens available for session 2 in the leaky bucket, where $\theta_i \in \Theta_i$.

It is not hard to see that the admissible actions for player 1 are limited by the service rate and those of player 2 by the number of tokens available in leaky buckets and the token generating rates. Since the server can serve only a fixed number of packets during each period, if there are more packets waiting in the switch buffers than it can serve, it is not possible for the server to transmit them all. Thus, the controller has to pick the packets to transmit in such a way that it minimizes the overall total payoff to the users. Similarly, player 2 attempts to maximize the total payoff from the controller by controlling the number of arriving packets while satisfying the leaky bucket constraint. The maximum number of packets that can arrive from each session at each period is limited by the number of tokens available in the leaky bucket plus the token generating rate for the session.

If the arriving traffic is shaped by the leaky bucket flow controller, Generalized Processor Sharing (GPS) can provide certain delay guarantee by setting ϕ_i correctly. The GPS service discipline provides delay guarantees by assigning the guaranteed minimum service rates to each input session. If we normalize ϕ_i by the sum of ϕ_i's and multiply it by the service rate, then it gives us the minimum guaranteed service rate for session i. Thus, regardless of what the other sessions do, session i will be served at least at the guaranteed minimum service rate. This is one way of protecting users from other malicious users that try to maximize their own utilities at other users' cost. For more details on GPS, refer to [11, 12, 13].

One problem with the GPS service discipline is that it uses a fluid model and assumes that more than one packet can be served simultaneously. Obviously this cannot be implemented in practice. Thus, instead of trying to implement GPS, one implements packet-by-packet Generalized Processor Sharing (PGPS), which is a non-preemptive modification of GPS. The maximum difference in the packet delay as well as the maximum backlog under GPS and PGPS is described in [12].

In our adaptive modification of packet-by-packet GPS, the controller decides which packet in the switch buffer to serve next according to the optimal strategy computed by the Shapley recursion method [14]. Unfortunately, these optimal strategies do not provide an upper bound on the delay that is as tight as under GPS. The decision of which next packet to serve depends on the state, ξ, alone. Simulation shows that our adaptive modification of PGPS indeed reduces the number of packets dropped whether sources have the same arrival rates or different arrival rates. The improvement in packet loss rate, however, comes at a price. The simulation reveals that the maximum delay under adaptive modification of PGPS is bigger than that under PGPS regardless of arrival rates.

3 Leaky Bucket Flow Control Scheme and the Virtual Backlog

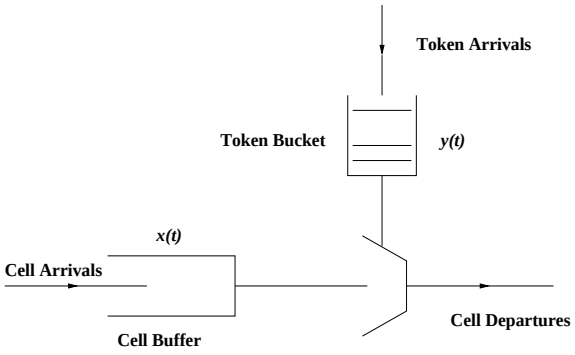

Figure 2: The leaky bucket scheme.

The basic idea behind the leaky bucket scheme is to regulate the admission of the arriving packets into the network. Each packet is allowed into the network only if there is a token available, and each packet consumes one token. The tokens are generated at a constant rate with constant interarrival times. The unused tokens get stored in a token bucket for each session. Also, tokens arriving when the bucket is full are lost, as are the packets arriving when the switch buffer is full.

This scheme has been found to be very effective in reducing the burstiness of arriving traffic. Anantharam and Konstantopoulos [4] proved the following result : Consider a stationary point process as bringing one unit of work with each point. Define a stationary point process to be less bursty than another such process if the steady state queue length in any single server queue working at rate strictly bigger than the total rate of arrival process, is statistically smaller for the former arrival process than for the latter. Then, for *any* stationary arrival process of offered traffic into a leaky bucket flow controller, if the token arrival rate is at least as big as the rate of offered traffic, the burstiness of the stationary departure process is monotonically increasing in the size of the token buffer. The degenerate case of infinite token bucket size corresponds to the case where there is no flow control at all. Thus, this result shows that the departure process from the leaky bucket is less bursty than the arrival process into the leaky bucket.

The departure process from the leaky bucket is upper bounded by an affine function of the length of the interval. If the size of the leaky bucket is C and the token generating rate is ρ, the total number of packets that can be admitted into the network by the leaky bucket over a time interval $[a, b]$ is upper bounded by $C + 1 + \rho(b - a)$. Much study has been done on this type of burstiness constrained traffic models starting with Cruz [6, 7].

This key observation suggests a new way of handling burstiness constrained flows because a recursively updatable statistic, *virtual backlog*, which is defined below, completely summarizes the past of a burstiness constrained flow. This allows one to formulate the problem of designing adaptive control strategies at network elements fed by burstiness constrained flows, using the theory of stochastic

games.

Suppose that the message flow $(a_n, n \geq 0)$ is (σ, ρ) constrained.

Definition 1 *We say that the message flow $(a_n, n \geq 0)$ has initial virtual backlog σ_0, if the flow obeys the constraints*

$$\sum_{0}^{n} a_k \leq \sigma_0 + \rho(n+1) \ for \ all \ n \geq 0. \tag{3}$$

The following lemma tells us how to represent the information gained by having observed the past of a (σ, ρ) constrained flow [1].

Lemma 1 *Let $(a_n, n \geq 0)$ be a (σ, ρ) constrained flow with initial virtual backlog σ_0. Suppose that a_0 is revealed. Then the information gained about $(a_n, n \geq 1)$ is exactly summarized by the statement that $(a_n, n \geq 1)$ is a (σ, ρ) constrained flow with initial virtual backlog σ_1, where*

$$\sigma_1 = min\{\sigma_0 + \rho - a_0, \sigma\}. \tag{4}$$

There are two parts to this statement.

(1) Every (σ, ρ) constrained flow with initial virtual backlog σ_1 is consistent with being the portion $(a_n, n \geq 1)$ of a flow $(a_n, n \geq 0)$ that is known to be (σ, ρ) constrained with initial virtual backlog σ_0 and has initial flow a_0.

(2) A (σ, ρ) constrained flow $(a_n, n \geq 0)$ with initial virtual backlog σ_0 and with initial flow a_0 results in a flow $(a_n, n \geq 1)$ that is (σ, ρ) constrained with initial virtual backlog σ_1.

We can consider a network element being fed by a number of burstiness constrained flows and implementing actions based on the past information. Assume that the *state* of the network element at time $n, n = 0, 1, \cdots$, is denoted by an element $\xi_n \in \Xi$. At time n the network element chooses a *control action* $u_n \in U$. The state of the network element evolves according to the incoming flows at time n and the control action chosen at time n, resulting in the evolution equation

$$\xi_{n+1} = f(\xi_n, u_n, \underline{a}_n) \tag{5}$$

If there are K sources with ith flow being (σ_i, ρ_i) constrained, $1 \leq i \leq K$, the domain of f is $\Xi \times U \times \sum_{i=1}^{K} [0, \sigma^i + \rho^i]$. A *randomized adapted control strategy* at the network element is a choice of a probability distribution on U at each time n as a function of $\xi_{[0,n]}, u_{[0,n-1]}, \underline{\sigma}_0$, and $\underline{a}_{[0,n-1]}$. Here $\underline{\sigma}_0$ is the vector of initial virtual backlogs of the flows.

4 Stochastic Games and Shapley Recursion

With the setting described in section 2 and 3, the problem of finding a good controller strategy can be formulated as a zero-sum stochastic game. We can define a function $c(\xi, u, \underline{a})$ to represent the cost for taking action u when the current state is ξ and the incoming flows are given by $\underline{a}$. This cost can be interpreted as the cost paid by the controller to the users. Let $0 < \beta < 1$ be a discount factor. Then, the problem can be formulated as one of minimizing the total discounted cost written as

$$\sum_{n=0}^{\infty} \beta^n c(\xi_n, u_n, \underline{a}_n) \tag{6}$$

The minimization is done over all possible randomized adapted control strategies of the controller and the sources, where a randomized adapted strategy is defined as a probability distribution on the set of actions available for the player. Since the source action space is convex, this is equivalent to a worst case formulation where the controller tries to minimize the overall discounted cost over all possible burstiness constrained source sequences.

Using the theory of stochastic games, the following theorem can be proved. For more details on Shapley recursion, refer to [14].

Theorem 4.1 *Suppose Ξ is a complete separable metric space and U is compact. Suppose f is continuous. Suppose there is a continuous nonnegative function g defined on Ξ having compact level sets, a polynomial $p(\cdot)$, and a constant $K < \infty$ such that, for all $\xi \in \Xi, u \in U$, and $\underline{a} \in \sum_{i=1}^{K} [0, \sigma^i + \rho^i]$*

(1) $\mid g(f(\xi, u, \underline{a})) - g(\xi) \mid \leq K.$

(2) $c(\xi, u, \underline{a}) \leq p(g(\xi)).$

Then the stochastic game above admits a continuous value function that is the unique fixed point of a Shapley recursion. Further, both the controller and the sources have optimal stationary randomized adapted strategies which depend only on the state of the game. These are respectively given by the outer extremizers in the min-max and the max-min forms of the Shapley recursion.

The optimal strategies for the controller in this formulation can be found by iterating the Shapley recursion. Further, in principle they can be implemented in *real time* since the entire past history can be summarized by the simple statistic, *virtual backlog*. For simulation, however, we implemented *simple strategies* that were calculated off-line before the simulation was run. We compared the values calculated from both randomized strategies and simple strategies. The difference in values from simple strategies are no more than three percent of the values calculated from the randomized strategies.

5 Discrete-time PGPS and Adaptive Modification

Since we are using a discrete time model, it is important to understand the difference between PGPS and DTPGPS. Under PGPS, if a packet arrives while the server is idle, then server immediately starts working on the packet. Under DTPGPS, however, the packet has to wait till the beginning of the next time slot as shown in Figure 3. Thus, the departure times of the packets may be different.

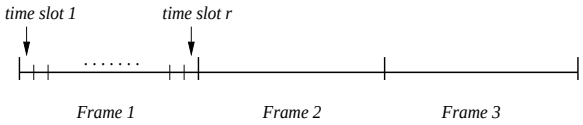

Figure 3: Discrete time PGPS.

In the simulation, each time slot corresponds to the amount of time necessary to transmit one byte, and the number of slots in each frame is equal to the total service rate in Mbps, which means that exactly r bytes get transmitted in each frame. Also note that the duration of each frame is 8 μs regardless of the service rate. For instance, if there are two sessions and the transmission rate of each session is 8 Mbps, then the duration of each slot will be 0.5 μs and there will be 16 slots in each frame.

In this setting even though arriving packets may get delayed, the stability condition is the same as under PGPS. Let $\tilde{F}_p$ denote the departure time of packet p under DTPGPS. We adopt the same convention for the arrival time that a packet is considered as having arrived only after the last bit of the packet has arrived. Since a packet has to wait till the next time slot, the difference in the departure times under PGPS and DTPGPS is upper bounded by the duration of one time slot, l μs. Also the difference in the session backlog is upper bounded by $r \cdot l$ bits. One thing to note is that the longest possible system busy period is the same under both schemes, which means that the maximum system busy period as well as the session busy period are bounded under DTPGPS. It is also clear that the session i maximum delay and backlog are upper bounded by $D_i^* + l$ and $Q_i^* + r \cdot l$, respectively, where D_i^* and Q_i^* are the session i maximum delay in μs and maximum backlog in bits under PGPS, respectively.

The adaptive modification of DTPGPS (ADTPGPS) is, however, rather different from DTPGPS. ADTPGPS allocates service rates in such a way that it minimizes

$$c(\xi_n, u_n, \underline{a}_n) + \beta \cdot v(\xi_{n+1}) \tag{7}$$

where $c(\xi_n, u_n, \underline{a}_n)$ is the amount paid to player 2 when actions u_n and $\underline{a}_n$ are chosen for player 1 and 2, respectively, at current state ξ_n, β is a discount factor, and $v(\xi_{n+1})$ is the value function in the state ξ_{n+1}. Given the current state ξ_n, and the actions u_n and $\underline{a}_n$ taken at time n, the state at $n+1$, ξ_{n+1}, can be calculated from equation (1).

Since ADTPGPS tries to redistribute the ϕ_i's based on the current state, there is no fixed guaranteed minimum service rate for each session and, thus, the analysis of the session maximum delay and backlog is much more complicated. One consequence of this is that the maximum session delay may be much bigger since all the service rate could be assigned to other sessions if there are more packets to be transmitted and tokens available for other sessions. For general input traffic, the analysis is very difficult since the value function depends not only on token generating rates and bucket sizes, but also on the switch buffer size of each session. The difficulty lies in understanding how the optimal strategies change depending on the number of backlogged packets and tokens available in the buckets. For instance, if the number of backlogged packets is small, then the optimal strategies calculated from the Shapley recursion method reveal that a higher service rate should be allocated to the session with more tokens available. If the switch buffers are almost full, a higher service rate should be given to the session with more backlogged packets, given that the difference in the number of tokens available is small enough.

The intuition behind ADTPGPS is that since dropping a packet has a relatively long effect, the cost for dropping a packet is bigger than that for holding a packet in the buffer. Thus, in order to minimize the total overall discounted cost, the switch should attempt to minimize the number of packets that the switch is forced to drop by assigning a higher service rate to the sessions with full buffers. Thus, this represents a tradeoff between tight upper bounds on the maximum session delays and minimizing the number of packets that get dropped.

6 Optimal Strategies

This section describes how the optimal strategies are computed and implemented. As mentioned before the optimal strategies are computed by iterating the Shapley recursion. Due to the size of the state space, we used a state space aggregation method to reduce the size of the state space we need to deal with [5]. For instance, if the token bucket size is T and the switch buffer size is B for each session, then the size of the state space is $(T+1)^I \cdot (B+1)^I$, where I is the number of input sessions. Thus, iterating the Shapley recursion for every state in the state space becomes impractical as the state space gets large. Obviously, since information is lost through state identification within an aggregate class, reducing the size of the state space through this approach leads to suboptimal solutions. The largest aggregate state space that was used for the computation of the optimal strategies is 26,896, where, for each session, the size of token buckets after state space aggregation is 41 and the size of switch buffer is 4. Let Ξ' denote the aggregate state space. For further simplicity, it is assumed that given that the current state is in some aggregate class $\xi' \in \Xi'$, every state in the aggregate class is

equally probable. This greatly simplifies the computation of the values associated with each aggregate state in Ξ' even though the strategies computed with this assumption may be suboptimal.

In order to justify the use of state space aggregation, we computed the optimal strategies and the values for each state with and without state space aggregation for a switch with token bucket size of 40 and switch buffer size of 10. The difference in the values we found was minimal. This could be due to the size of the switch. However, this justifies the use of state space aggregation for our purposes since the size of the state space of the switch we implemented for the simulation is not much bigger.

For the simulation, rather than implementing the randomized strategies, we used pure strategies for the simplicity of implementation. In order to make sure that the differences in the values associated with each state yielded by randomized strategies and pure strategies are not too big, we computed the strategies and the values for comparison. The difference in the values was no greater than three percent, and moreover, for most states the difference was smaller than one percent. This justifies the use of pure strategies in the place of randomized strategies.

In the simulation the holding cost for a packet is set to be the delay experienced by the packet. Since the cost for each dropped packet is relatively big compared to the cost for holding a packet in the switch buffer, the optimal strategies reveal that the controller should attempt to minimize the total payoff to player 2 by reducing the number of packet drops. This can be easily seen when one session almost fills up the switch buffer and is about to drop packets while the other session does not have nearly as many backlogged packets. The controller assigns most of its service rates to the session with many more backlogged packets. This is intuitively satisfying since it makes sense to serve the packets from the session with an almost full switch buffer in order to prevent packet drops. It is also easy to see how the number of tokens available for each session affects the controller strategy. For instance, if session 1 has many more tokens in the token buckets than session 2 while they have about the same number of packets in the switch buffer waiting for service, then session 1 is much more likely to have a packet drop. Thus, the controller allocates more service rate to session 1 than to session 2 in an attempt to reduce the number of dropped packets. These intuitions are verified by the analysis of the fluid model.

We also computed the optimal strategies for the case where the cost of holding a packet from session 1 in the switch buffer is twice of that of session 2, although we did not run the simulation for this case. Intuitively what the controller should do is to assign all its service rate to session 1 if there are not enough tokens available in the session 2 token bucket to cause any packet drop, since it is better to serve session 1 packet and keep session 2 packets waiting given that there is no possibility of dropping session 2 packets. This is indeed what the optimal strategies reveal. However, when there are enough tokens available for session 2 to cause packets to drop, the controller assigns some of the

service rate to session 2 to prevent the packets from being dropped. This becomes more apparent when the session 2 switch buffer gets almost full and there are more than enough tokens available to cause packet drops. Since the cost associated with packet drop is still bigger than holding a packet in the buffer, the controller attempts to serve session 2 packets as long as there are not many session 1 packets backlogged and tokens available.

7 Fluid Model

In this section, we describe the fluid model and investigate the structure of the optimal strategies of the players. Suppose that the service rate is sufficiently high and the packet size is rather small as in ATM networks, then the switch can be approximated by a fluid model. First, we will describe the model in section 7.1, and then attempt to characterize the optimal strategies of the players.

7.1 Model

There are I ≥ 1 users sharing the switch with a service rate of $r > 0$. Let $I = \{1, 2, \ldots, \mathrm{I}\}$ be the set of users. The goal of the users is to maximize the payoffs from the controller, and that of the controller is to minimize the payoffs to the users. We first describe the action spaces for the players. At each decision time $n \geq 0$, player 2, who represents the users, controls when the fluid will be admitted into the network between time n and $n+1$, provided that the amount of fluid being admitted into the network from each user i at any given instance does not exceed the amount of available token. In other words, player 2 controls the shape of the fluid being admitted into the network as shown in Figure 4. Player 1, the controller, decides at each decision time n how much service rate will be given to each user during the period $[n, n+1)$. Under these assumptions, the action space of player 1 is convex. Let Φ_i denote the strategy space of player i.

Note that, in the fluid model, unlike in the original model where the number of packets admitted into the network by each user is always an integer value, player 2 can admit any amount of fluid into the switch at any given time as long as it satisfies the leaky bucket constraints.

We now define the cost function for the controller. In the previous sections, the cost function has two parts, a cost for holding packets in the buffer and a penalty for dropping packets. In this section, we assume that the cost for holding fluid in the buffer is negligible compared to the penalty for dropping fluid, so the cost function consists only of penalty for dropping fluid. The total discounted cost is given by

$$c(\phi_1, \phi_2, \xi) = \int_0^\infty \beta^t p(t, \phi_1, \phi_2, \xi)\, dt, \tag{8}$$

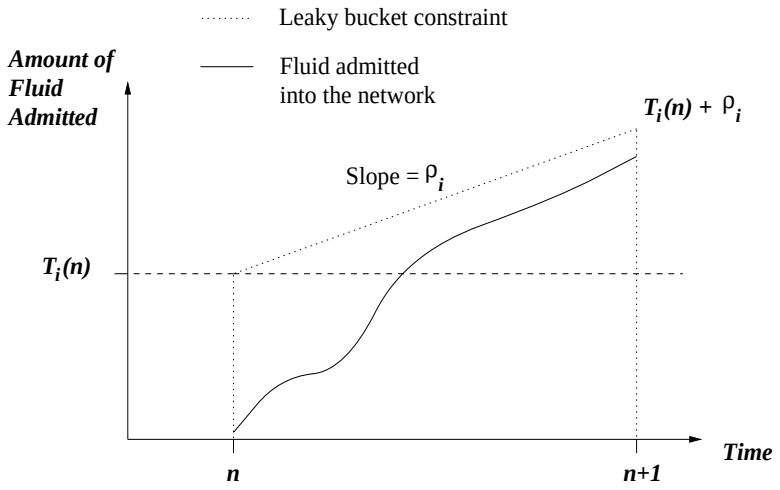

Figure 4: Action of player 2.

where $\beta < 1$ is the discount factor, ϕ_i is player i's strategy, and $p(t, \phi_1, \phi_2, \xi)$ is the rate at which fluid is dropped at time t, when the initial condition is ξ.

7.2 Optimal Strategies

Let us first define the optimal strategies for the players. Any $\tilde{\phi_1} \in \Phi_1$ such that for all $\phi_1 \in \Phi_1$ and all $\xi \in \Xi$

$$max_{\phi_2} c(\phi_1, \phi_2, \xi) \geq max_{\phi_2} c(\tilde{\phi_1}, \phi_2, \xi) \tag{9}$$

i.e., $max_{\phi_2} c(\tilde{\phi_1}, \phi_2, \xi) = min_{\phi_1} max_{\phi_2} c(\phi_1, \phi_2, \xi)$, is optimal for the controller. Similarly, any $\tilde{\phi_2} \in \Phi_2$ such that for all $\phi_2 \in \Phi_2$ and all $\xi \in \Xi$

$$min_{\phi_1} c(\phi_1, \phi_2, \xi) \leq min_{\phi_1} c(\phi_1, \tilde{\phi}_2, \xi) \tag{10}$$

i.e., $min_{\phi_1} c(\phi_1, \tilde{\phi}_2, \xi) = max_{\phi_2} min_{\phi_1} c(\phi_1, \phi_2, \xi)$, is optimal for player 2. In this section we investigate the optimal strategies for the players when the discount factor β approaches one.

Let T_i, B_i, and ρ_i be the token bucket size, switch buffer size, and token generating rate of session $i \in I$, respectively. Throughout this section we assume that $T_i > B_i$ for each user i. If there are users with token bucket size smaller than or equal to the switch buffer size, then assuming that the stability condition holds, i.e., $\sum_{j \in I} \rho_j \leq r$, the controller can assign to those users fixed service rates equal to their token generating rates and eliminate the possibility of fluid drop from those sessions. In this case, the payoff to player 2 from these users will be zero.

Let us first describe the *wait-and-dump* procedure that will be used in the analysis of the optimal strategies of the players. Given any initial state, over a long period, each user can force the switch to drop $\frac{T_i - B_i}{T_i}$ of the total fluid it admits into the network as follows. Suppose that each user always waits

till its token bucket is full and then dumps as much fluid as possible, namely T_i amount of fluid, into the network. Since the switch can hold only up to B_i amount of fluid in the buffer, $T_i - B_i$ amount of fluid will be dropped. Therefore, over a long period the users can force the switch to drop $\frac{T_i - B_i}{T_i}$ of the total fluid by repeating the above wait-and-dump procedure.

Being aware of this, in order to minimize the total payoffs to player 2, the controller should attempt not to drop any more fluid than what it is absolutely forced to. This means that if the stability condition holds, if possible, over a long period the controller should not drop any more than $\frac{T_i - B_i}{T_i}$ of the total fluid from each user i. One example of a scheduling algorithm that achieves this is *Rate Proportional Processor Sharing* (RPPS), where the service rate assigned to each user i is $r \cdot \frac{\rho_i}{\sum_j \rho_j}$, because $r \cdot \frac{\rho_i}{\sum_j \rho_j} \geq \rho_i$. It is shown later that RPPS is one of optimal strategies of the controller under the stability assumption when the discount factor is sufficiently close to one.

7.3 Main Results

In this subsection, we identify a class of controller strategies that can be easily written in terms of the current state, which are optimal as the discount factor gets sufficiently close to one. Suppose that the current state is $\xi \in \Xi$ at time n. Let $T_i(n), 0 \leq T_i(n) \leq T_i$, denote the amount of token available in user i's token bucket in the current state $\xi \in \Xi$ at time $n \geq 0$, and $B_i(n), 0 \leq B_i(n) \leq B_i$, the amount of user i's backlogged fluid in its switch buffer. Let $\beta < 1$ be the discount factor and $F_i = \frac{T_i}{\rho_i}$, the amount of time required for user i to fill up the empty token bucket. We assume that F_i is much larger than 1, i.e., $T_i \gg \rho_i$. The service rate assigned to user i' by the controller at time n is denoted by $r_i(n) \geq 0$.

Let Φ_1^0 be the set of player 1's strategies that satisfy the following conditions for each user $i \in I$ and all $\xi \in \Xi$:

(A) If $\frac{T_i(n) - (B_i - B_i(n))}{T_i(n)} < \frac{T_i - B_i}{T_i}$ and $t \triangleq \frac{T_i - T_i(n)}{\rho_i} < 1$,
then $r_i(n) > \frac{\rho_i B_i}{T_i}$.

(B) If $\frac{T_i(n) - (B_i - B_i(n))}{T_i(n)} < \frac{T_i - B_i}{T_i}$ and $t \geq 1$,
then $r_i(n) > max\{\frac{\rho_i B_i}{T_i} + \frac{T_i B_i(n) + B_i T_i(n) - B_i T_i}{T_i}, 0\}$.

(C) If $\frac{T_i(n) - (B_i - B_i(n))}{T_i(n)} = \frac{T_i - B_i}{T_i}$,
then $r_i(n) > \frac{\rho_i B_i}{T_i}$.

(D) If $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} > \frac{T_i-B_i}{T_i}$,
then $r_i(n) > \frac{\rho_i B_i}{T_i}$.

and ϕ_2^0 be the strategy of player 2 satisfying the following for each user $i \in I$ and all $\xi \in \Xi$:

(A') If $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} < \frac{T_i-B_i}{T_i}$ and $t < 1$,
then user i waits till $n+t$ and uses up the token when the token bucket gets full.

(B') If $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} < \frac{T_i-B_i}{T_i}$ and $t \geq 1$,
then user i waits till $n+1$.

(C') If $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} = \frac{T_i-B_i}{T_i}$,
then user i uses up all its token at time n.

(D') If $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} > \frac{T_i-B_i}{T_i}$,
then user i uses up all its token at time n.

Note that there exists at least one controller strategy that satisfies (A) - (D) if the stability condition holds. The following claim states that Φ_1^0 is a set of optimal strategies for player 1 and ϕ_2^0 is an optimal strategy for player 2 for β sufficiently close to one.

Claim Φ_1^0 is a set of optimal strategies for player 1 and ϕ_2^0 is an optimal strategy for player 2 as β approaches one.

Proof: The proof of the claim is given in Appendix A. ■

An obvious observation one can make from the claim is the decoupling among the users. This may seem counterintuitive at first. The intuition is, however, that as the discount factor β approaches one, the users pay attention to the percentage of fluid that gets discarded over a long period. This means that even if it is possible for some user i to drop certain amount of fluid at the current time n, it may want to wait to save more token to increase the percentage of the fluid that gets dropped later, depending on the current state.

8 Simulation

We have run the simulation with the network traffic traces taken at Bellcore. Since these are IP packets, we fragmented them into packets of 100 bytes assuming that transmitters send the packets at 100 Mbps. The average traffic load of the samples we have used is about 8 Mbps for each session for the first three parts, and 5 Mbps and 7 Mbps for the first session and the second session, respectively, for the last part.

For the simulation, we pre-computed appropriate values for token bucket sizes and token generating rates of input sources before we ran the simulation. While we could easily find the average arrival rates of input sessions, we had to run many simulations with different token bucket sizes to find the appropriate values for the bucket sizes. We will not discuss the issue of translating the burstiness of traffic into the token bucket size here, and assume that there is a mechanism that translates one to the other.

8.1 Fixed Token Bucket Size

For the first simulation we used the fragmented traffic samples with average load of about 8 Mbps for each session, i.e., 16 Mbps from both sessions. We used about 250,000 packets for each simulation. The first part of the simulation examines how much better ADTPGPS performs compared to DTPGPS when the average load from each session is about the same. We first fixed the token bucket size, the token generating rate, and the switch buffer size for both sessions, and then increased the service rates for both sessions at the same rate at different switch buffer sizes. Figure 5 and 6 show that ADTPGPS consistently does better than DTPGPS in term of number of packets dropped. However, note that the maximum delay under ADTPGPS is considerably greater than under DTPGPS as shown in Table 1. As mentioned before, this is due to the fact that ADTPGPS reassigns the service rate to each session based on the current state and does not provide any guaranteed minimum service rate to either session. Thus, if one session has many backlogged packets in the buffer and keeps admitting more packets into the network while the other session has only a few backlogged packets for some period, then the backlogged packets from the latter session will receive very little service as long as the first session keeps admitting packets at the token generating rate, and those packets will experience longer delays than they would under DTPGPS. This suggests the possibility that there may be a way of assigning a minimum service rate for each session and optimally assigning the residual service rate based on the current state. This represents the tradeoff between the number of packets dropped and the maximum delay.

First note that each time slot corresponds to 0.5 μs and each frame delay is 8 μs (0.5×16 μs).

Thus, a delay of 380 frames equals a delay of 3.04 ms (380 * 8 μs). The token generating rate is 10^4 token/sec, and the service rate is in Mbps.

Switch Buffer Size = 30 (packets)					
	DTPGPS			ADTPGPS	
	service rate	ses1	ses2	ses1	ses2
Token Bucket = 50	8.0	380	379	524	557
	8.5	356	336		
	9.0				
Token Bucket = 60	8.0	380	379	524	557
	8.5	356	336	415	373
	9.0	270	259		
Token Bucket = 70	8.0	380	379	524	582
	8.5	356	336		
	9.0	270	259		
Token Bucket = 80	8.0	380	379	528	582
	8.5	356	336	453	373
	9.0	236	224		
Token Bucket = 90	8.0	380	379	528	582
	8.5	356	336	453	373
	9.0	270	259		
Token Bucket = 100	8.0	380	379	528	582
	8.5	356	336	453	373
	9.0	270	259		

Table 1. Maximum Packet Delay under DTPGPS and ADTPGPS (in frames)

8.2 Variable Token Bucket Size

For the second part of the simulation we fixed the switch buffer sizes and increased the token bucket sizes to see how the size of token bucket affects the performance of ADTPGPS compared to that of DTPGPS. Figure 7, and 8 show that ADTPGPS still performs better than DTPGPS if we are only interested in the number of packets dropped, which is how we formulated the problem, i.e., the cost function for a packet is linear with the delay the packet experiences, with a heavy penalty for being

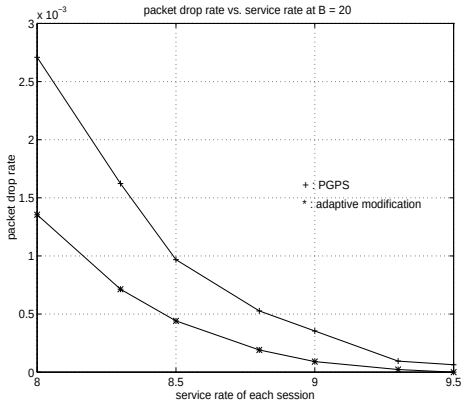

Figure 5: Packet Drop Rate of PGPS and Adaptive Modification at B = 20.

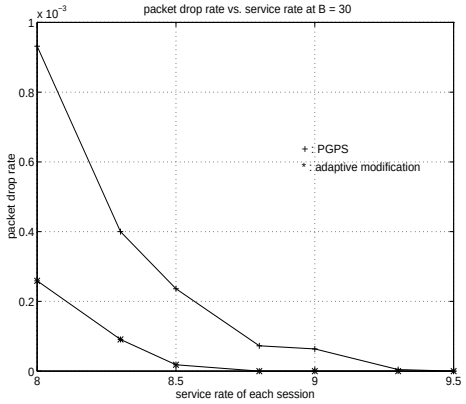

Figure 6: Packet Drop Rate of PGPS and Adaptive Modification at B = 30.

dropped. Again, Table 1 shows that the maximum packet delay under ADTPGPS is still considerably bigger than that under DTPGPS for the same reason as explained before.

8.3 Effect of Token Bucket Size

In this subsection, we will take a look at how the token bucket size affects the packet loss rate. Anantharam and Konstantopoulos [3] show that the burstiness is an increasing function of the token bucket size. Thus, we naturally expect the packet loss rate to increase with the token bucket size, and this is indeed the case as shown in Figure 9 and 10.

8.4 Different Arrival Rates

The last part of the simulation examines the case where two sessions have different arrival rates. We want to show that even when sessions have different arrival rates, ADTPGPS still incurs smaller overall cost by reducing the packet drop rate at the cost of increased maximum delay. The first session has an arrival rate of 5 Mbps and the second session 7 Mbps. We used about 60,000 packets for the first

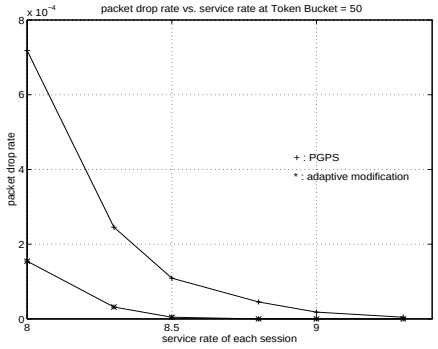

Figure 7: Packet Drop Rate of PGPS and Adaptive Modification at Token Bucket = 50.

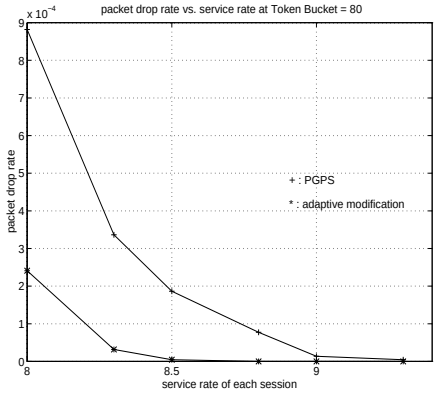

Figure 8: Packet Drop Rate of PGPS and Adaptive Modification at Token Bucket = 60.

session and 100,000 packets for the second session during the simulation. We chose the part of the network traffic trace that has low average arrival rate but is rather bursty for the first session. The second session, although it has a higher mean arrival rate, is much less bursty compared to the first session. This can be seen from Figure 11 and 12. Since the first session is much more bursty than the second session, the switch drops many more packets from the first session, even though it has a lower mean arrival rate. Also, note that the difference between DTPGPS and ADTPGPS is not as big as in the case where both sessions have the same arrival rates. One of the reasons is that although the switch attempts to assign as much service rate to the first session as possible when the session 1 switch buffer gets full at the cost of dropping some of the packets from the second session, it cannot serve all the packets that come in batches without dropping many of them. This can be seen from the fact that the number of session 1 lost packets is reduced at the cost of an increase in the number of lost packets from session 2. These results are summarized in Table 2 and 3.

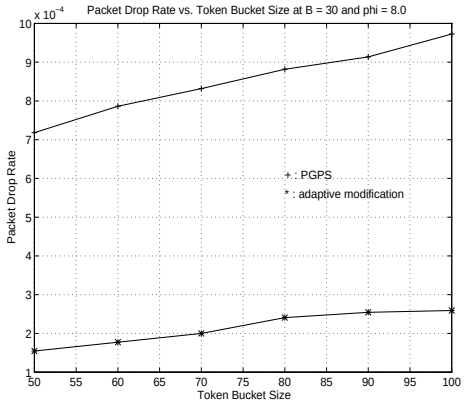

Figure 9: Packet Drop Rate of PGPS and Adaptive Modification at Token Bucket = 100 and Phi = 8.0.

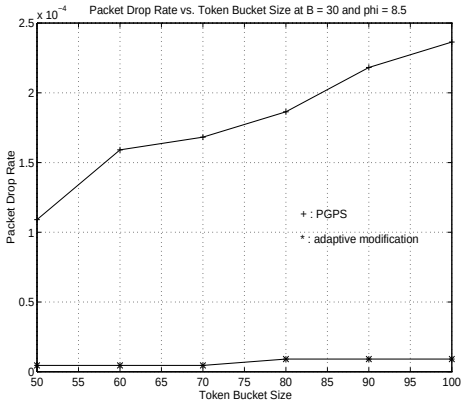

Figure 10: Packet Drop Rate of PGPS and Adaptive Modification at Token Bucket = 100 and Phi = 8.5.

TK : session 1 = 100, session 2 = 150					
	DTPGPS			ADTPGPS	
	service rate	ses1	ses2	ses1	ses2
Buffer Size = 30	7, 8	297	73	120	74
	7.5, 8.5	138	22	62	30
	8, 9	61	4	49	13
Buffer Size = 40	7, 8	113	36	64	33
	7.5, 8.5	49	4	42	3
	8, 9	24	0	10	6

Table 2. Number of Packets Dropped for DTPGPS and ADTPGPS

TB : session 1 = 100, session 2 = 150					
	DTPGPS			ADTPGPS	
	service rate	ses1	ses2	ses1	ses2
Buffer	7, 8	439	382	683	507
Size	7.5, 8.5	405	343	626	410
= 30	8, 9	382	246	568	384
Buffer	7, 8	578	506	904	540
Size	7.5, 8.5	540	478	840	485
=40	8, 9	504	373	668	463

Table 3. Maximum Packet Delay for DTPGPS and ADTPGPS

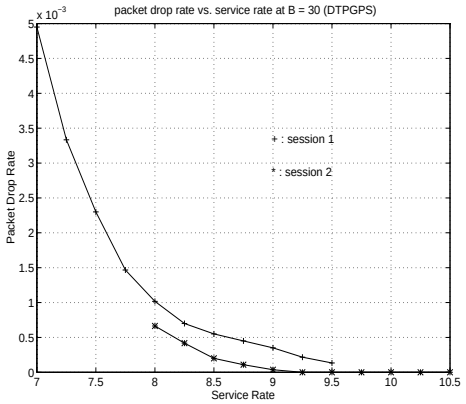

Figure 11: Packet Drop Rate of DTPGPS with B=30.

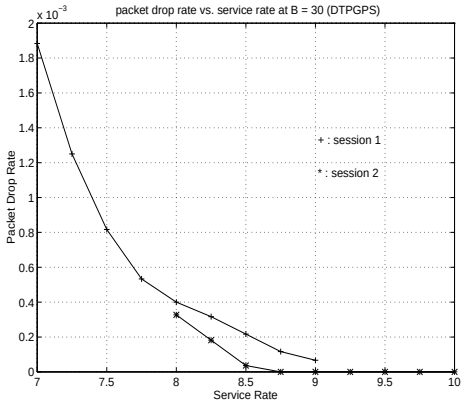

Figure 12: Packet Drop Rate of ADTPGPS with B=30.

9 Conclusion

As part of the efforts to find an efficient multiplexing scheme that can handle traffic from various types of sources, we have investigated the possibility of improving the performance of a switch by modeling the scheduling problem as a zero-sum stochastic game between the controller and the users. We found that an adaptive modification of Discrete-time PGPS (ADTPGPS) performs better than Discrete-time PGPS (DTPGPS) in terms of the packet loss rate whether the sessions have same arrival rates or different arrival rates. However, one can see that the difference is much bigger when the arriving sessions have similar arrival rates with comparable burstiness than when one source is much more bursty than the other even though the mean arrival rate may be smaller. This suggests that the performance of ADTPGPS depends on the burstiness of the arrival processes.

The improvement in the packet loss rate, however, comes at a price : the maximum delay can be considerably bigger under ADTPGPS, which is a consequence of ADTPGPS not providing any guaranteed minimum service rates to the users. Moreover, ADTPGPS is much more difficult to analyze due to the complexity of the optimal strategies for the switch.

References

[1] V. Anantharam, "An approach to the design of high speed networks for bursty traffic," *Proceedings of the 32nd IEEE Conference on Decision and Control,* New York, NY. IEEE 1993, Vol. 2, pp. 1678-9.

[2] V. Anantharam, "Bandwidth allocation to burstiness constrained flows," *IEEE Information Theory Workshop,* June 1992.

[3] V. Anantharam and T. Konstantopoulos, "Optimal flow control schemes that regulate the burstiness of traffic," *IEEE/ACM Transactions on Networking,* Aug. 1995, Vol. 3, pp. 423-32.

[4] V. Anantharam and T. Konstantopoulos, "Burst reduction properties of the leaky bucket flow control scheme in ATM networks," *IEEE Transactions on Communications,* Dec. 1994, Vol. 42, pp. 3085-9.

[5] D. Bertsekas, *Dynamic programming and stochastic control* New York : Academic Press, 1976.

[6] R. Cruz, "A calculus for network delay, Part 1: Network elements in isolation," *IEEE Trans. Inform. Theory,* Jan 1991, Vol. 37, pp. 114-121.

[7] R. Cruz, "A calculus for network delay, Part 2: Network analysis," *IEEE Trans. Inform. Theory,* Jan. 1991, Vol. 37, pp. 121-141.

[8] D. Ferrari and D. Verma, "A scheme for real-time communication services," *IEEE J. Selected Areas in Commun.*, Vol. 8, pp. 368-379, Apr. 1990.

[9] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE J. Selected Areas in Commun.*, Vol. 9, pp. 126579, Oct. 1991.

[10] L. Kleinrock, *Queueing System. Vol. 2: Computer Applications.* New York: Wiley 1976.

[11] A. Parekh, "A generalized processor sharing approach to flow control in integrated services network," Ph.D. dissertation, MIT, Feb. 1992.

[12] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control-The single node case," in *Proc. INFOCOM '92*, 1992

[13] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case," in *Proc. INFOCOM '93,* San Francisco, CA, Mar. 1993, pp. 521-530.

[14] L. Shapley, "Stochastic Games," *Proc. N.A.S.,* Vol. 39, 1953, pp. 1095-1100.

[15] H. Zhang "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. of the IEEE,* Oct. 1995, Vol. 83, pp. 1374-96.

[16] L. Zhang "Virtual Clock: A new traffic control algorithm for packet switching networks", *Proc. ACM SIGCOMM '90*, Philadephia, PA, Sept. 1990, pp. 19-29.

A Proof of Claim in Section 7.3

We first describe the outline of the proof. Suppose that we show that, for all $\phi_1^0 \in \Phi_1^0$ and $\xi \in \Xi$,

$$max_{\phi_2} c(\phi_1^0, \phi_2, \xi) = c(\phi_1^0, \phi_2^0, \xi), \tag{11}$$

and

$$c(\phi_1^0, \phi_2^0, \xi) = min_{\phi_1} c(\phi_1, \phi_2^0, \xi). \tag{12}$$

Since we already know that the game has a value, i.e.,

$$max_{\phi_2} min_{\phi_1} c(\phi_1, \phi_2, \xi) = min_{\phi_1} max_{\phi_2} c(\phi_1, \phi_2, \xi) = c(\xi), \tag{13}$$

equations (11) and (12) imply that, for all $\phi_1^0 \in \Phi_1^0$ and $\xi \in \Xi$,

$$\begin{aligned} min_{\phi_1} max_{\phi_2} c(\phi_1, \phi_2, \xi) &\leq max_{\phi_2} c(\phi_1^0, \phi_2, \xi) \\ &= c(\phi_1^0, \phi_2^0, \xi) \\ &= min_{\phi_1} c(\phi_1, \phi_2^0, \xi) \\ &\leq max_{\phi_2} min_{\phi_1} c(\phi_1, \phi_2, \xi) \end{aligned} \tag{14}$$

and that ϕ_1^0 and ϕ_2^0 are optimal strategies for players 1 and 2, respectively. Therefore, it suffices to show that (11) and (12) are true in order to prove the claim.

(i) $max_{\phi_2} c(\phi_1^0, \phi_2, \xi) = c(\phi_1^0, \phi_2^0, \xi)$

We prove (11) by showing that, for all $\xi \in \Xi$, $\phi_1^0 \in \Phi_1^0$, and $\phi_2 \in \Phi_2$

$$c(\phi_1^0, \phi_2^0, \xi) \geq c(\phi_1^0, \phi_2, \xi) \tag{15}$$

This part of the proof is organized in the following way. First, assume that player 1 adopts a strategy that satisfies (A)-(D), but the service rate assigned to each user i by the controller at any period n is no higher than ρ_i. Then, we may consider each user i in isolation. In other words, we consider I ≥ 1 new games, in each of which there is only one user i and the service rate of the controller is limited to at most ρ_i. We investigate the optimal strategy for user i when the controller adopts a strategy that satisfies $(A) - (D)$. We will show that in this new game the user i's strategy φ_i^0 that satisfies $(A') - (D')$ is an optimal strategy. We now argue that if we limit the service rate assigned to user i to at most ρ_i, when users act cooperatively, they cannot increase their payoffs from the controller compared to the case where they are considered in isolation. Since the controller never assigns to a user a service rate higher than its token generating rate, the action of one user does not affect what controller does to other users. Therefore, in this case each user playing the strategy given by $(A') - (D')$ gives rise to an optimal strategy for player 2. Last, we consider the original model where the restriction on player 1's strategy is removed. We will show that distributing the extra available service rate over the users after removing the restriction, does not change the payoffs to the users. This can be explained by observing that the extra service rate affects the amount of backlogged fluid, but not the amount of discarded fluid. Obviously, when extra service rate is distributed among the users, users cannot increase their payoffs. Therefore, player 2's optimal strategy remains the same.

We now assume that the strategy adopted by the controller satisfies $(A) - (D)$ and, in addition, that the service rate assigned to any user at any period is at most its token generating rate. Then, we isolate each user and investigate its optimal strategy in the new game with a controller with a service rate of at most ρ_i. The controller in this case, however, does not need to allocate all of its service rate,

i.e., has a variable service rate. Let Ψ_i be the set of controller i strategies in isolation with maximum available service rate of ρ_i, Ψ_i^0 the set of controller i strategies that satisfy (A)-(D), and Ξ_i the state space of the new game. Suppose that the current state is $\xi_i \in \Xi_i$ at period n. Let $T_i(n)$ and $B_i(n)$ denote the amount of token available and backlogged fluid, respectively, in current state ξ at time n. Let $t = \frac{T_i - T_i(n)}{\rho_i}$, the amount of time it takes to fill up the token bucket from time n. We will consider the following four possible cases and show that a user i's strategy φ_i^0 that satisfies $(A')-(D')$, is optimal.

(a) $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} < \frac{T_i-B_i}{T_i}$ and $t < 1$, (A1)

(b) $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} < \frac{T_i-B_i}{T_i}$ and $t \geq 1$, (A2)

(c) $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} = \frac{T_i-B_i}{T_i}$, (A3)

(d) $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} > \frac{T_i-B_i}{T_i}$, (A4)

We will show that no strategy of user i yields a higher payoff than φ_i^0 by investigating what user i should do in each of the above four cases (a)-(d). The intuition behind the proof is as follows. The users already know that they can make the controller drop $\frac{T_i-B_i}{T_i}$ of the total fluid over a long period. Hence, each user i should look at the current state at time n and see if it can force the controller to drop at least $\frac{T_i-B_i}{T_i}$ of $T_i(n) + B_i(n)$, i.e., $T_i(n) + B_i(n) - B_i \geq (T_i(n) + B_i(n))\frac{(T_i-B_i)}{T_i}$. If so, then the user should use up all its token at time n. If not, the user should wait till it is true or the token bucket is full.

(a) First note that if (A1) holds at time n, then $t > 0$. Suppose that user i waits to collect the token for future use. Then, with the assumption $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} < \frac{T_i-B_i}{T_i}$, we can show that at time $(n+t)^-$, there is no backlog in the buffer as follows.

$$\begin{aligned} B_i(n) &< \frac{B_i(T_i - T_i(n))}{T_i} \\ &= \frac{B_i\rho_i}{T_i}\frac{T_i - T_i(n)}{\rho_i} \\ &= \rho_i\frac{B_i}{T_i}t. \end{aligned} \tag{16}$$

We now show that if user i saves the token for future use and does not admit any fluid, then the assumption (A1) continues to hold for all $n + \tau, 0 \leq \tau < t$. If $B_i(n + \tau) = 0$ for some $0 \leq \tau < t$, then (A1) holds trivially. Thus, we need to consider only $\tau \in [0, \frac{B_i(n)}{r_i(n)}]$. Let $\kappa = \frac{B_i(n)}{r_i(n)} < t$. We can show

after a little algebra that, for all $\tau \in (0, \kappa]$,

$$\begin{aligned} & \frac{T_i(n+\tau) - (B_i - B_i(n+\tau))}{T_i(n+\tau)} \\ < & \frac{T_i(n) + \rho_i\tau - B_i + B_i(n) - \frac{\rho_i B_i}{T_i}\tau}{T_i(n) + \rho_i\tau} \\ = & \frac{T_iT_i(n) + T_i\rho_i\tau - T_iB_i + T_iB_i(n) - \rho_iB_i\tau}{T_i(T_i(n) + \rho_i\tau)} \end{aligned} \tag{17}$$

and

$$\begin{aligned} & \frac{T_i(n+\tau) - (B_i - B_i(n+\tau))}{T_i(n+\tau)} - \frac{T_i - B_i}{T_i} \\ < & \frac{T_iT_i(n) + T_i\rho_i\tau - T_iB_i + T_iB_i(n) - \rho_iB_i\tau}{T_i(T_i(n) + \rho_i\tau)} - \frac{T_i - B_i}{T_i} \\ = & \frac{T_iT_i(n) + T_i\rho_i\tau - T_iB_i + T_iB_i(n) - \rho_iB_i\tau}{T_i(T_i(n) + \rho_i\tau)} \\ & + \frac{-T_iT_i(n) - T_i\rho_i\tau + B_iT_i(n) + B_i\rho_i\tau}{T_i(T_i(n) + \rho_i\tau)} \\ = & \frac{T_iB_i(n) + B_iT_i(n) - T_iB_i}{T_i(T_i(n) + \rho_i\tau)} \\ < & 0. \end{aligned}$$

where the last inequality follows from the assumption $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} < \frac{T_i-B_i}{T_i}$, which implies that $T_iB_i > T_iB_i(n) + T_i(n)B_i$.

Suppose that the service rate to user i is fixed at ρ_i. Then, when user i starts with an empty token bucket. user i should always play wait-and-dump in order to maximize its payoff. Hence, if the controller adopts an optimal strategy, user i cannot receive any higher payoff than the payoff it receives by playing wait-and-dump because optimal strategies by definition should not allow a higher payoff than fixing the service rate at ρ_i. Therefore, we can assume that once its token bucket becomes empty, user i's expected future payoff is the payoff it receives when it plays wait-and-dump. The question for the users now becomes choosing when to first empty the bucket given the current state. We will show that when (A1) is true at time n, user i receives a higher payoff (i) if it plays wait-and-dump than (ii) when it uses up all its token at time n and plays wait-and-dump afterwards. As argued above, this implies that if (A1) holds at time n, then user i should always wait till the token bucket becomes full because (A1) continues to hold for all $n + \tau$, $0 \le \tau < t$, if the controller adopts a strategy adopts a strategy in Ψ_i^0. Note that if $T_i(n) = 0$, then (i) and (ii) are equivalent. Thus, we will assume that $T_i(n) > 0$.

In case (ii), user i receives a total discounted payoff of

$$[T_i(n) + B_i(n) - B_i]^+ + \beta^{F_i}(T_i - B_i) + \beta^{2F_i}(T_i - B_i) + \cdots. \tag{18}$$

In case (i) user i receives

$$\beta^t(T_i - B_i) + \beta^{t+F_i}(T_i - B_i) + \cdots. \tag{19}$$

When the discount factor approaches one, we will show that

$$\begin{aligned} &\beta^t(T_i - B_i) + \beta^{t+F_i}(T_i - B_i) + \cdots] \\ &-([T_i(n) + B_i(n) - B_i]^+ + \beta^{F_i}(T_i - B_i) + \beta^{2F_i}(T_i - B_i) + \cdots) \\ &> 0. \end{aligned} \tag{20}$$

After a little algebra, equation (20) becomes

$$\begin{aligned} &\beta^t(T_i - B_i) + \beta^{t+F_i}\frac{T_i - B_i}{1-\beta^{F_i}} - ([T_i(n) + B_i(n) - B_i]^+ + \beta^{F_i}\frac{T_i - B_i}{1-\beta^{F_i}}) \\ = \;& \beta^t(T_i - B_i) - [T_i(n) + B_i(n) - B_i]^+ - (\beta^{F_i} - \beta^{t+F_i})\frac{T_i - B_i}{1-\beta^{F_i}} \end{aligned} \tag{21}$$

As $\beta \to 1$,

$$\frac{\beta^{F_i} - \beta^{t+F_i}}{1-\beta_i^F} \to \frac{t}{F_i} \tag{22}$$

and $\beta^t \to 1$. Thus, as $\beta \to 1$, equation (21) has the limit

$$\begin{aligned} &(T_i - B_i) - [T_i(n) + B_i(n) - B_i]^+ - \frac{t}{F_i}(T_i - B_i) \\ = \;& (1 - \frac{t}{F_i})(T_i - B_i) - [T_i(n) + B_i(n) - B_i]^+. \end{aligned} \tag{23}$$

Since $\frac{t}{F_i} = \frac{T_i - T_i(n)}{T_i}$, equation (23) is equal to

$$\begin{aligned} &(1 - \frac{T_i - T_i(n)}{T_i})(T_i - B_i) - [T_i(n) + B_i(n) - B_i]^+ \\ = \;& \frac{T_i(n)}{T_i}(T_i - B_i) - [T_i(n) + B_i(n) - B_i]^+. \end{aligned} \tag{24}$$

If $[T_i(n) + B_i(n) - B_i]^+ = 0$, then obviously (24) is greater than 0 from the assumptions $T_i > B_i$ and $T_i(n) > 0$. If $[T_i(n) + B_i(n) - B_i]^+ > 0$, then (24) is equal to

$$\begin{aligned} &T_i(n) - \frac{T_i(n)}{T_i}B_i - T_i(n) - B_i(n) + B_i \\ = \;& \frac{T_iB_i - B_i(n)T_i - T_i(n)B_i}{T_i}. \end{aligned} \tag{25}$$

From (A1), the numerator in (25) is greater than 0, and the claim follows.

The above argument shows that if (A1) holds at any time, then user i should play wait-and-dump if the controller strategy satisfies the conditions given in (A)-(D), for (A1) continues to hold till the token bucket is full. Therefore, user i's best strategy against any $\psi_i^0 \in \Psi_i^0$ is to play wait-and-dump.

We can also show that if user i plays wait-and-dump after it uses up all its token at $n+t$, then (A2) holds at time $n+1$ from

$$\begin{aligned}
& \frac{T_i(n+1) - B_i + B_i(n)}{T_i(n+1)} - \frac{T_i - B_i}{T_i} \\
< & \frac{\rho_i(1-t) - B_i + (B_i - \rho_i \frac{B_i}{T_i}(1-t)}{\rho_i(1-t))} - \frac{T_i - B_i}{T_i} \\
= & \frac{\rho_i(1-t)T_i - \rho_i B_i(1-t) - \rho_i(1-t)T_i + \rho_i B_i(1-t)}{T_i \rho_i(1-t)} \\
= & \ 0 \qquad (26)
\end{aligned}$$

and that T_i is assumed to be much larger than ρ_i.

(b) Assumption $\frac{T_i(n)-(B_i-B_i(n))}{T_i(n)} < \frac{T_i-B_i}{T_i}$ implies that $T_iB_i > B_i(n)T_i + B_iT_i(n)$. We first consider the case where $max\{\frac{\rho_i B_i}{T_i} + \frac{T_iB_i(n)+B_iT_i(n)-B_iT_i}{T_i}, 0\} = 0$. This implies that $\rho_i B_i \leq B_iT_i - B_iT_i(n) - B_i(n)T_i$. Thus, for all $\tau < 1 \leq \frac{B_iT_i-B_iT_i(n)-B_i(n)T_i}{\rho_i B_i}$, we have

$$\begin{aligned}
& \frac{T_i(n) + \rho_i\tau + B_i(n) - B_i}{T_i(n) + \rho_i\tau} - \frac{T_i - B_i}{T_i} \\
= & \frac{T_iT_i(n) + T_i\rho_i\tau + T_iB_i(n) - T_iB_i}{T_i(T_i(n) + \rho_i\tau)} - \frac{T_iT_i(n) - T_i(n)B_i + \rho_i\tau T_i - \rho_i\tau B_i}{T_i(T_i(n) + \rho_i\tau)} \\
= & \frac{T_iB_i(n) - T_iB_i + T_i(n)B_i + \rho_i\tau B_i}{T_i(T_i(n) + \rho_i\tau)} \\
< & \frac{T_iB_i(n) - T_iB_i + T_i(n)B_i + \rho_i B_i \frac{T_iB_i - T_iB_i(n) - B_iT_i(n)}{\rho_i B_i}}{T_i(T_i(n) + \rho_i\tau)} \\
= & \ 0.
\end{aligned}$$

Hence, if user i saves token for the future, then for all $\tau < 1$

$$\frac{T_i(n+\tau) + B_i(n) - B_i}{T_i(n+\tau)} < \frac{T_i - B_i}{T_i}. \qquad (27)$$

Moreover, if $t > 1$, then one can easily show that $\frac{T_i(n+1)-B_i+B_i(n+1)}{T_i(n+1)} < \frac{T_i-B_i}{T_i}$.

We now consider the case where $max\{\frac{\rho_i B_i}{T_i} + \frac{T_iB_i(n)+B_iT_i(n)-B_iT_i}{T_i}, 0\} > 0$. From the assumption, we have $T_i - T_i(n) \geq \rho_i$. We first show that, for all $\tau < 1$,

$$\begin{aligned}
& B_i(n) - \frac{\rho_i B_i + T_iB_i(n) + B_iT_i(n) - B_iT_i}{T_i}\tau \\
= & \frac{B_i(n)T_i - (\rho_i B_i + T_iB_i(n) + B_iT_i(n) - B_iT_i)\tau}{T_i} \\
= & \frac{B_i(n)T_i(1-\tau) + (-\rho_i B_i + B_i(T_i - T_i(n))\tau}{T_i} \\
\geq & \frac{B_i(n)T_i(1-\tau) + (-\rho_i B_i + B_i\rho_i)\tau}{T_i} \\
= & \frac{B_i(n)T_i(1-\tau)}{T_i} \\
\geq & \ 0. \qquad (28)
\end{aligned}$$

Note that we have an equality when $\tau = 1$ and $t = 1$. Suppose that user i saves the token for the future. Then, as proved in (a), we can show that, for all $\tau \in [0,1)$, either (A1) or (A2) holds, i.e.,

$$\frac{T_i(n+\tau) - (B_i - B_i(n+\tau))}{T_i(n+\tau)} < \frac{T_i - B_i}{T_i}, \tag{29}$$

as follows.

$$\begin{aligned}
& \frac{T_i(n+\tau) - (B_i - B_i(n+\tau))}{T_i(n+\tau)} \\
< \; & \frac{T_i(n) + \rho_i\tau - B_i + B_i(n) - \frac{\rho_i B_i + T_i B_i(n) + B_i T_i(n) - B_i T_i}{T_i}\tau}{T_i(n) + \rho_i\tau} \\
= \; & \frac{T_i T_i(n) + T_i\rho_i\tau - T_i B_i + T_i B_i(n) - (\rho_i B_i + T_i B_i(n) + B_i T_i(n) - B_i T_i)\tau}{T_i(T_i(n) + \rho_i\tau)} \\
= \; & \frac{T_i T_i(n) + T_i\rho_i\tau - \rho_i B_i\tau - B_i T_i(n)\tau - T_i(1-\tau)(B_i - B_i(n))}{T_i(T_i(n) + \rho_i\tau)},
\end{aligned} \tag{30}$$

where the first inequality follows from (28), and

$$\begin{aligned}
& \frac{T_i(n+\tau) - (B_i - B_i(n+\tau))}{T_i(n+\tau)} - \frac{T_i - B_i}{T_i} \\
< \; & \frac{T_i T_i(n) + T_i\rho_i\tau - \rho_i B_i\tau - B_i T_i(n)\tau - T_i(1-\tau)(B_i - B_i(n))}{T_i(T_i(n) + \rho_i\tau)} - \frac{T_i - B_i}{T_i} \\
= \; & \frac{T_i T_i(n) + T_i\rho_i\tau - \rho_i B_i\tau - B_i T_i(n)\tau - T_i(1-\tau)(B_i - B_i(n))}{T_i(T_i(n) + \rho_i\tau)} \\
& + \frac{-T_i T_i(n) - T_i\rho_i\tau + B_i T_i(n) + B_i\rho_i\tau}{T_i(T_i(n) + \rho_i\tau)} \\
= \; & \frac{B_i T_i(n)(1-\tau) - T_i(1-\tau)(B_i - B_i(n))}{T_i(T_i(n) + \rho_i\tau)} \\
= \; & \frac{(1-\tau)(B_i T_i(n) + T_i B_i(n) - T_i B_i)}{T_i(T_i(n) + \rho_i\tau)} \\
< \; & 0.
\end{aligned} \tag{31}$$

The last inequality follows from (A2), which implies $T_i B_i > T_i B_i(n) + T_i(n) B_i$. One can also prove that if $t > 1$, then $\frac{T_i(n+1) - B_i + B_i(n+1)}{T_i(n+1)} < \frac{T_i - B_i}{T_i}$.

Now we show that if (A2) holds at time n, then user i should play wait-and-dump in order to maximize the payoff by comparing the two cases mentioned in (a). First, we consider the case where user i uses up its token at time n. In this case, user i receives a total discounted payoff of

$$[T_i(n) + B_i(n) - B_i]^+ + \beta^{F_i}(T_i - B_i) + \beta^{2F_i}(T_i - B_i) + \cdots. \tag{32}$$

If user i plays wait-and-dump, then it receives

$$\beta^t(T_i - B_i) + \beta^{t+F_i}(T_i - B_i) + \cdots. \tag{33}$$

The remaining steps are identical to the ones shown in (a), and we omit them here. Hence, we have

$$\begin{aligned} & \beta^t(T_i - B_i) + \beta^{t+F_i}(T_i - B_i) + \cdots \\ > \quad & [T_i(n) + B_i(n) - B_i]^+ + \beta^{F_i}(T_i - B_i) + \beta^{2F_i}(T_i - B_i) + \cdots. \end{aligned} \tag{34}$$

Since (A1) or (A2) continues to hold till the token bucket is full, user i should play wait-and-dump in order to maximize its payoff.

(c) Suppose $t = \frac{T_i - T_i(n)}{\rho_i} = 0$, i.e., $T_i = T_i(n)$. If (A3) holds, we must have $B_i(n) = 0$. Then, clearly user i should use up its token at time n. Hence, we assume that $t > 0$. This implies that $B_i(n) > 0$. Suppose $t < 1$. Then, $B_i(n+t) = 0$ if user i does not admit fluid over the period $[n, n+t]$, because

$$t = \frac{T_i - T_i(n)}{\rho_i} = \frac{T_i \cdot B_i(T_i - T_i(n))}{T_i \cdot \rho_i B_i} = \frac{B_i(n)}{\frac{\rho_i B_i}{T_i}}, \tag{35}$$

where the last equality follows from $B_i(n) = \frac{B_i(T_i - T_i(n))}{T_i}$, which comes from (A3). Thus, the amount of time it takes to fill up the token bucket is no less than that of emptying the switch buffer. Similarly as in case (a) and (b), if (A3) holds and user i saves the token for the future, i.e., does not admit any fluid over period $[n, n+\tau]$, then $\frac{T_i(n+\tau) - (B_i - B_i(n+\tau))}{T_i(n+\tau))} < \frac{T_i - B_i}{T_i}$ for $\tau, 0 < \tau < \min\{t,1\}$ as follows.

$$\begin{aligned} & \frac{T_i(n+\tau) - (B_i - B_i(n+\tau))}{T_i(n+\tau)} \\ < \quad & \frac{T_i(n) + \rho_i\tau - B_i + B_i(n) - \rho_i\frac{B_i}{T_i}\tau}{T_i(n) + \rho_i\tau} \\ = \quad & \frac{T_iT_i(n) + T_i\rho_i\tau - T_iB_i + T_iB_i(n) - \rho_iB_i\tau}{T_i(T_i(n) + \rho_i\tau)} \\ = \quad & \frac{(T_i - B_i)T_i(n) + T_i\rho_i\tau - \rho_iB_i\tau}{T_i(T_i(n) + \rho_i\tau)}, \end{aligned} \tag{36}$$

where the last equality follows from $T_iB_i = T_i(n)B_i + B_i(n)T_i$, which comes from (A3). Hence,

$$\begin{aligned} & \frac{T_i(n+\tau) - (B_i - B_i(n+\tau))}{T_i(n+\tau)} - \frac{T_i - B_i}{T_i} \\ < \quad & \frac{(T_i - B_i)T_i(n) + T_i\rho_i\tau - \rho_iB_i\tau}{T_i(T_i(n) + \rho_i\tau)} - \frac{T_i - B_i}{T_i} \\ = \quad & \frac{(T_i - B_i)T_i(n) + T_i\rho_i\tau - \rho_iB_i\tau - T_iT_i(n) + T_i(n)B_i - T_i\rho_i\tau + B_i\rho_i\tau}{T_i(T_i(n) + \rho_i\tau)} \\ = \quad & \frac{(T_i - B_i)T_i(n) + T_i(n)(B_i - T_i)}{T_i(T_i(n) + \rho_i\tau)} \\ = \quad & 0. \end{aligned} \tag{37}$$

Therefore, for all $0 < \tau < min\{1, t\}$,

$$\frac{T_i(n+\tau) - (B_i - B_i(n+\tau))}{T_i(n+\tau)} < \frac{T_i - B_i}{T_i}. \tag{38}$$

Furthermore, if $t > 1$, then (38) holds for $\tau = 1$.

We now show that if (A3) holds at any time, user i cannot increase its total payoff by waiting to save more token. In other words, we show that for all $\tau, 0 \leq \tau < \min\{t,1\}$, as the discount factor β goes to one

$$\begin{aligned}
& (T_i(n) + B_i(n) - B_i) + \beta^{F_i} \frac{T_i - B_i}{1 - \beta^{F_i}} \\
& -(\beta^{\tau}(T_i(n) + \rho_i \tau + [B_i(n) - r_i(n)\tau]^{+} - B_i) + \beta^{\tau + F_i} \frac{T_i - B_i}{1 - \beta^{F_i}}) \\
> \quad & (T_i(n) + B_i(n) - B_i) + \beta^{F_i} \frac{T_i - B_i}{1 - \beta^{F_i}} \\
& -(\beta^{\tau}(T_i(n) + \rho_i \tau + B_i(n) - \rho \frac{B_i}{T_i}\tau - B_i) + \beta^{\tau + F_i} \frac{T_i - B_i}{1 - \beta^{F_i}}) \\
\geq \quad & 0. \qquad (39)
\end{aligned}$$

The first equality follows from (35) and condition in (c). From equation (22), we know that $\frac{\beta^{F_i} - \beta^{\tau + F_i}}{1 - \beta^{F_i}} \rightarrow \frac{\tau}{F_i}$ and $\beta^{\tau} \rightarrow 1$ as $\beta \rightarrow 1$. Thus, as $\beta \rightarrow 1$,

$$\begin{aligned}
& (T_i(n) + B_i(n) - B_i) + \beta^{F_i} \frac{T_i - B_i}{1 - \beta^{F_i}} \\
& -(\beta^{\tau}(T_i(n) + \rho_i \tau + B_i(n) - \rho \frac{B_i}{T_i}\tau - B_i) + \beta^{\tau + F_i} \frac{T_i - B_i}{1 - \beta^{F_i}}) \\
\rightarrow \quad & \frac{\tau}{F_i}(T_i - B_i) - \rho_i \tau (1 - \frac{B_i}{T_i}) \text{ as } \beta \rightarrow 1, \\
= \quad & \frac{\tau \rho_i}{T_i}(T_i - B_i) - \rho_i \tau (1 - \frac{B_i}{T_i}) \\
= \quad & \rho_i \tau \frac{(T_i - B_i) - (T_i - B_i)}{T_i} \\
= \quad & 0 \qquad (40)
\end{aligned}$$

From equation (39) and (40), we can see that if $r_i(n) > \rho_i \frac{B_i}{T_i}$, no other strategies for user i yield a higher payoff than φ_i^0.

(d) We show that as in case (c), user i cannot increase its payoff by waiting to save more token. First suppose that $t > 0$. We show that, for all $0 \leq \tau < \min\{t,1\}$,

$$\begin{aligned}
& (T_i(n) + B_i(n) - B_i) + \beta^{F_i} \frac{T_i - B_i}{1 - \beta^{F_i}} \\
& -(\beta^{\tau}(T_i(n) + \rho_i \tau + [B_i(n) - r_i(n)\tau]^{+} - B_i) + \beta^{\tau + F_i} \frac{T_i - B_i}{1 - \beta^{F_i}}) \\
> \quad & (T_i(n) + B_i(n) - B_i) + \beta^{F_i} \frac{T_i - B_i}{1 - \beta^{F_i}} \\
& -(\beta^{\tau}(T_i(n) + \rho_i \tau + B_i(n) - \rho \frac{B_i}{T_i}\tau - B_i) + \beta^{\tau + F_i} \frac{T_i - B_i}{1 - \beta^{F_i}}) \\
\geq \quad & 0. \qquad (41)
\end{aligned}$$

Since it is in the interest of user i to drop the fluid at the earlier possible time if it drops the same amount of fluid, it suffices to compares φ_i^0 only to the wait-and-dump policy after waiting τ amount of time. The first inequality follows from the assumption $B_i(n) > \frac{B_i(T_i - T_i(n))}{T_i}$. However, equation (41) is same as equation (39). Thus, from equations (39) and (40), equation (41) follows. If $t = 0$, then it is easy to see that user i should use up all its token. Hence, no other user i strategy yields higher payoffs than playing φ_i^0.

The above proves that φ_i^0 is indeed an optimal strategy for user i when user i is considered in isolation. As argued before, this also proves that φ_i^0's constitute an optimal strategy for player 2 in the original game when player 1's strategy is not allowed to assign a service rate higher than ρ_i to user i at any period. We now argue that these φ_i^0's give rise to an optimal strategy for player 2 even when this restriction is removed. This can be seen from the following. First, with increased service rates assigned to the users, obviously the users cannot increase their payoffs from the case with the restriction on player 1's strategy. Hence, if possible, the best users can do is to keep the same payoff as in the previous case with the restriction in place. It is easy to show that when the restriction is removed the users' payoffs remain the same. If either (A1) or (A2) holds for user i in current state ξ, then any controller strategy that satisfies the conditions in $(A) - (D)$ forces user i to play wait-and-dump and exactly $T_i - B_i$ amount of fluid is discarded when user i uses up all of its token. Hence, increasing the service rate to user i does not reduce the payoff to the user because same amount of token gets dropped regardless of the controller strategy. If (A3) or (A4) holds in the current state ξ at time n, then user i uses up all available token at time n and plays wait-and-dump afterwards. Clearly, increasing the service rate to user i does not reduce the amount of fluid discarded at time n and the future wait-and-dump payoffs stay the same. Hence, this proves that ϕ_2^0 is indeed optimal for player 2 against any $\phi_1^0 \in \Phi_1^0$, i.e., for all $\xi \in \Xi, \phi_1^0 \in \Phi_1^0$, and $\phi_2 \in \Phi_2$

$$c(\phi_1^0, \phi_2^0, \xi) \geq c(\phi_1^0, \phi_2, \xi), \tag{42}$$

and this proves equation (11).

(ii) $c(\phi_1^0, \phi_2^0, \xi) = min_{\phi_1} c(\phi_1, \phi_2^0, \xi)$

We now proceed to prove equation (12) by showing that, for all $\phi_1 \in \Phi_1$ and $\xi \in \Xi$,

$$c(\phi_1^0, \phi_2^0, \xi) \leq c(\phi_1, \phi_2^0, \xi). \tag{43}$$

First we argue that, for all $\tilde{\phi}_1$ and $\bar{\phi}_1 \in \Phi_1^0$,

$$c(\tilde{\phi}_1, \phi_2^0, \xi) = c(\bar{\phi}_1, \phi_2^0, \xi). \tag{44}$$

Then, we show that for all $\phi_1^0 \in \Phi_1^0, \phi_1 \in \Phi_1 \setminus \Phi_1^0$, and $\xi \in \Xi$,

$$c(\phi_1^0, \phi_2^0, \xi) \leq c(\phi_1, \phi_2^0, \xi), \tag{45}$$

which proves equation (12).

Let us first prove equation (44)

$$c(\tilde{\phi}_1, \phi_2^0, \xi) = c(\bar{\phi}_1, \phi_2^0, \xi) \tag{46}$$

for all $\tilde{\phi}_1$ and $\bar{\phi}_1 \in \Phi_1^0$. This follows from the previous calculations in part (i). If either (A1) or (A2) holds in current state for some user i, then under all $\phi_1^0 \in \Phi_1^0$, from previous calculations user i's buffer is always empty when user i uses up its token according to ϕ_2^0. Therefore, the payoff user i receives is the same under all $\phi_1^0 \in \Phi_1^0$ because same amount of fluid is discarded at all times. If (A3) or (A4) holds for user i in current state at time n, then it uses up its token at time n and plays wait-and-dump afterwards. In this case, no matter what the controller does at time n, the payoff to user i for the discarded fluid at time n remains constant, namely $\beta^n(T_i(n) + B_i(n) - B_i)$. Furthermore, after time n, whenever user i uses up its token according to ϕ_2^0, its buffer is empty under all $\phi_1^0 \in \Phi_1^0$. Therefore, user i's payoff is the same under all $\phi_1^0 \in \Phi_1^0$ if player 2 plays ϕ_2^0.

We proceed to show that

$$c(\phi_1^0, \phi_2^0, \xi) \leq c(\phi_1, \phi_2^0, \xi) \tag{47}$$

for all $\phi_1^0 \in \Phi_1^0, \phi_1 \in \Phi_1 \setminus \Phi_1^0$, and $\xi \in \Xi$. This can be argued as follows. From (39), (40), and (41), we know that when (A3) or (A4) holds for some user i, if it plays according to ϕ_2^0 it does not receive a lower payoff than wait-and-dump payoff. Hence, for all users for which (A1) or (A2) holds, the controller should force them to play wait-and-dump and not allow either (A3) or (A4) to hold at a later decision time until the token bucket gets filled. In other words, if (A1) or (A2) holds for some user i in the current state ξ at time n, then the controller should ensure that when user i plays according to ϕ_2^0, it waits till the token bucket gets full. Note that this is exactly what the strategies in Φ_1^0 do as shown before. Therefore, any player 1 strategy that does not satisfy (A) or (B) cannot reduce the payoffs to the users. Now for those users for which (A3) or (A4) is true in current state, the controller should make sure that after they use up their token at time n they wait till the next time token bucket becomes full and that the buffer is empty when the token bucket is filled. Again, this is what all $\phi_1^0 \in \Phi_1^0$ do. Hence, no other player 1 strategy can reduce player 2's payoffs when player 2 adopts ϕ_2^0.

The above argument together with (13) and (14) proves the claim that Φ_1^0 is a set of player 1's optimal strategies and ϕ_2^0 is a player 2's optimal strategy when the discount factor is sufficiently large.