MODELLING AND OPTIMIZATION OF COMPUTER NETWORK TRAFFIC CONTROLLERS

N. U. AHMED, BO LI, AND LUIS OROZCO-BARBOSA

Received 28 February 2005

During the past years, there has been increasing interest in the design and development of network traffic controllers capable of ensuring the QoS requirements of a wide range of applications. In this paper, we construct a dynamic model for the token-bucket algorithm: a traffic controller widely used in various QoS-aware protocol architectures. Based on our previous work, we use a system approach to develop a formal model of the traffic controller. This model serves as a basis to formally specify and evaluate the operation of the token-bucket algorithm. Then we develop an optimization algorithm based on a dynamic programming and genetic algorithm approach. We conduct an extensive campaign of numerical experiments allowing us to gain insight on the operation of the controller and evaluate the benefits of using a genetic algorithm approach to speed up the optimization process. Our results show that the use of the genetic algorithm proves particularly useful in reducing the computation time required to optimize the operation of a system consisting of multiple token-bucket-regulated sources.

1. Introduction

During the past years, there has been increasing interest in the definition of simple but yet effective traffic control schemes able to ensure the grade of service required by a wide variety of applications. By effective, it is understood that the control mechanism should also ensure a proper level of utilization of the network resources, a fundamental condition to ensure the profitability of operation.

In this paper, we focus our efforts on the study of the token bucket (TB): a traffic controller widely used in various QoS-aware protocol architectures [12]. A token bucket is associated with a traffic source in order to regulate the rate at which the source may send its traffic into the network. In its simplest form, a token bucket is characterized by a bucket whose capacity is expressed in tokens with a token corresponding to a number of bytes. A token generation rate determines the rate at which tokens are generated and placed in the bucket. If upon a packet arrival, enough tokens are available, the token bucket is decreased accordingly to the packet size, the packet is then classified as belonging to the conforming traffic and sent into the network. Otherwise, the packet is discarded,

DOI: 10.1155/MPE.2005.617

that is, no buffers are provided to temporally hold the incoming packets. When the source rate is low or inactive, new tokens can be saved up in the bucket. If the bucket becomes full, the new generated tokens are simply discarded.

The token-bucket mechanism has been studied widely in the literature. In recent years, various authors have used the token-bucket model to characterize the traffic generated by different types of sources, such as video and voice. In [7], Lombaedo et al. have used the token-bucket model to characterize the traffic generated by prerecorded MPEG video. The main objective of their work has been to provide an analytical methodology to compute the traffic specifications used by RSVP: the signaling protocols of the IntServ protocol architecture. In [11], Procissi et al. have also used the token bucket to characterize traffic exhibiting long-range dependence (LRD). Their numerical results show the effectiveness of their analytical approach for the proper sizing of the token-bucket parameters for LRD traffic. In [3], Bruni and Scoglio have also used the token-bucket model to optimize the traffic generation rate of video sources when satisfying the token-bucket controls. In [4], Bruno et al. have estimated the token-bucket parameters for voice over IP traffic. In [13], Tang and Tai have conducted a network traffic characterization using the token-bucket algorithm. Besides the bufferless token-bucket model, the authors have also analyzed the operation of a token bucket using a buffer to hold the incoming source packets.

In this paper, we undertake the modelling and optimization of the token-bucket algorithm. Our token-bucket model differs from most previous studies in two major points. First, rather than characterizing a particular traffic type, we focus on the modelling of the token-bucket algorithm. Therefore we make no assumptions whatsoever on the nature of the traffic sources. Most studies in the past [1, 12, 15, 18] have been conducted assuming a particular type of application (e.g., video or voice) or traffic statistics (e.g., ON/OFF, LRD dependence). Second, rather than focusing on a single token bucket, our study on optimization looks into a system consisting of multiple token buckets. The interaction among multiple token-bucket sources is considered by explicitly including in our models the use of a shared resource, that is, a multiplexor. Furthermore, we propose the use of a dynamic programming/genetic algorithm approach for the optimization process. The use of the genetic algorithm has proved useful to considerably reduce the search process, particularly as the number of token buckets increases.

The paper is organized as follows. Section 2 introduces the basic notation and the dynamic model of the token bucket including the shared resources. In Section 3, we define the objective function and provide the details of the dynamic programming/genetic algorithm approach proposed for the optimization of a network consisting of multiple token buckets. In Section 4, we conduct a set of numerical experiments to illustrate the effectiveness of the proposed approach. In Section 5, we analyze the benefits of the proposed methodology in terms of computation time reduction.

2. Traffic model and system model

In our study, we present a dynamic model based on the basic philosophy of IP networks. This model includes some improvement of the previous system model developed in [1]. Here we present a new methodology to solve the optimal control problem in computer

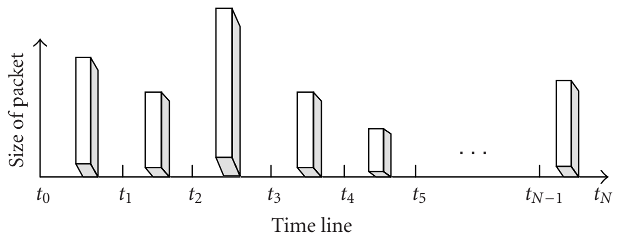


Figure 2.1. Traffic model.

communication network. Before describing the system model, let us define the following symbols which are used throughout the paper.

(1) $\{x \wedge y\} = \text{Min}\{x,y\}$, $\{x \vee y\} = \text{Max}\{x,y\}$, $x,y \in \mathbb{R}$.

(2) For $X,Y \in \mathbb{R}^n$ the components of the vector $Z \equiv \{X \wedge Y\}$ are given by $z_i \equiv x_i \wedge y_i$, $i = 1,2,\dots,n$, where $X = \{x_i,\ i = 1,2,\dots,n\}$ and $Y = \{y_i,\ i = 1,2,\dots,n\}$. Similarly for $Z \equiv X \vee Y$ we write $z_i \equiv x_i \vee y_i$.

(3)

$$I(s) = \begin{cases} 1 & \text{if the predicate } S \text{ is true,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

2.1. Traffic model. There are various types of services which are carried out in the network, such as data, voice, and video. The traffic generated by these services can be the constant bit rate (CBR) traffic or variable bit rate (VBR) traffic. The traffic model considered here remains the same as in the previous work [1]. We build up our traffic model with VBR which is shown on Figure 2.1. The incoming traffic is denoted by $\{v(t_{k-1}),\ k = 1,2,\dots,N\}$. Since during each time interval $[t_{k-1},t_k)$ at most one packet may arrive, $v(t_{k-1})$ then represents the length or the size of the packet.

The VBR traffic can be very bursty; therefore the bandwidth allocation to the traffic is hard to determine. The peak rate allocation would result in the waste of network resources and may even lead to congestion, while the mean rate or lower than the mean rate allocation would result in losses and the QoS (quality of service) may not be guaranteed.

2.2. System model. We consider that the system involves n traffic sources, and that each source is assigned to a TB that controls the traffic rate. Figure 2.2 shows the system model. The TB model is modified to take into account the capacity c_i, of the input link i, where $i = 1,2,\dots,n$. This is the link between the token bucket and the multiplexor. We use c to denote the vector of capacities of the input links, where $c = (c_i,\ i = 1,2,\dots,n)$.

The system model is formally given by the following vector difference equation:

$$\begin{aligned} \rho(t_k) &= \rho(t_{k-1}) + \{u(t_{k-1}) \wedge [T - \rho(t_{k-1})]\} \\ &\quad - v(t_{k-1}) I(v(t_{k-1}) \le \{[\rho(t_{k-1}) + [u(t_{k-1}) \wedge (T - \rho(t_{k-1}))]] \wedge c\tau\}), \end{aligned} \tag{2.2}$$

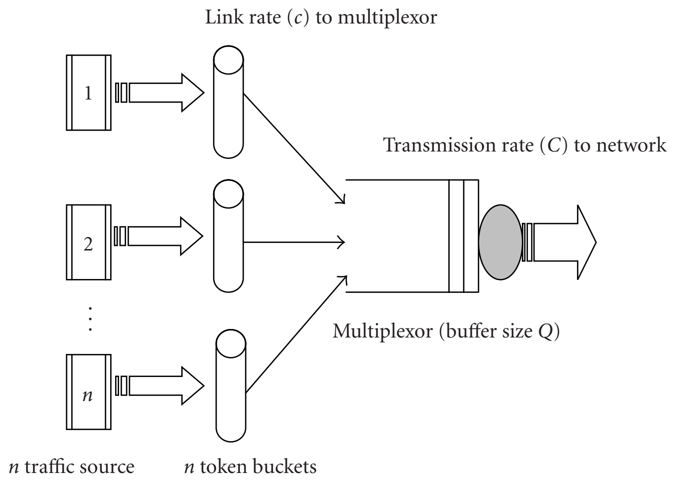

Link rate (c) to multiplexor
1
2
n
Transmission rate (C) to network
Multiplexor (buffer size Q)
n traffic source
n token buckets

Figure 2.2. System model.

where the vector $\rho(t_k) = (\rho_i(t_k),\ i = 1,2,\dots,n)$ denotes the status of token occupancy of all the token buckets at time t_k, that is, $\rho_i(t_k)$ is the number of tokens in the ith TB at time t_k, $u(t_{k-1})$ is also a vector $(u_i(t_{k-1}),\ i = 1,2,\dots,n)$, which denotes the incoming tokens to each TB at the start of the time interval $[t_{k-1}, t_k)$. The vector $T = (T_i,\ i = 1,2,\dots,n)$ denotes the size (maximum capacity) of all the TBs, $\nu(t_{k-1}) = (\nu_i(t_{k-1}),\ i = 1,2,\dots,n)$ is a vector denoting the size of the arriving packets from each source during the time interval $[t_{k-1}, t_k)$, and τ is the length of the time interval.

The conforming traffic (i.e., the traffic matching with the available tokens in the token banks) from each of the TBs is then given by

$$g_i(t_k) = \nu_i(t_k) I\{\nu_i(t_k) \le [\rho_i(t_k) + (u_i(t_k) \wedge (T_i - \rho_i(t_k)))] \wedge c_i\tau\}. \tag{2.3}$$

Clearly this means that the nonconforming traffic, denoted by $r_i(t_k)$, is given by

$$r_i(t_k) = \nu_i(t_k) - g_i(t_k). \tag{2.4}$$

In this study, we assume that the (traffic) sources share the same multiplexor in an access node at the edge of the network. The multiplexor collects all the sources in its buffer before they are launched on to the outgoing link depending on the available bandwidth or capacity at the time. Thus, to complete the dynamic model of the whole system, one must also include the temporal variation of the queue at the multiplexor. For simplicity we may assume that the bandwidth of the link is piecewise constant on each of the intervals $[t_{k-1}, t_k)$ and it is denoted by $C(t_{k-1})$, and the multiplexor has finite buffer sizes Q. Let the size of the queue, $q(t_k)$, waiting for service at the multiplexor at time t_k denote the state of the multiplexor. The dynamics of the multiplexor queue is then given by the following (information) balance equation:

$$q(t_k) = \{[q(t_{k-1}) - C(t_{k-1})\tau] \vee 0\} \\ + \left\{\left[\sum_{i=1}^{n} g_i(t_{k-1})\right] \wedge [Q - ([q(t_{k-1}) - C(t_{k-1})\tau] \vee 0)]\right\}. \tag{2.5}$$

The first term on the right-hand side of the expression (2.5) describes the leftover traffic in the queue at time t_k after the traffic has been injected into the network. The second term represents all the conforming traffic accepted by the multiplexor during the same period of time. If the network capacity is large enough to serve all the traffic in the queue at time t_{k-1}, the state of the queue or the multiplexor is given by all the incoming traffic accepted. But in case the network cannot serve all the waiting traffic, the unserved traffic is stored in the queue. If the buffer with size Q does not have enough space to store all the unserved traffic, the excess is discarded.

Because of the limitation of the buffer size and the output link capacity, some traffic may be dropped at the multiplexor. Thus the traffic losses at the multiplexor is described by

$$L(t_k) = \left[\sum_{i=1}^{n} g_i(t_k)\right] - \left\{\left[\sum_{i=1}^{n} g_i(t_k)\right] \wedge [Q - ([q(t_k) - C\tau] \vee 0)]\right\}. \tag{2.6}$$

The term $\{[\sum_{i=1}^{n} g_i(t_k)] \wedge [Q - ([q(t_k) - C\tau] \vee 0)]\}$ represents the part of conforming traffic accepted by the multiplexor. If the buffer space, Q, is large enough to accept all the conforming traffic, no multiplexor losses would occur at time t_k. Otherwise, some part of the conforming traffic must be dropped.

According to (2.2) and (2.5), we may formally write the abstract model which will be used later in our algorithm and computation as follows. Let $X = \binom{\rho}{q} \in \mathbb{R}^{n+1}$ denote the state of the system consisting of n token banks and one multiplexor. Then the system dynamics can be described compactly by the following system of difference equations:

$$X(k+1) = F(k, X(k), u(k)), \quad k = 0, 1, 2, \ldots, \tag{2.7}$$

where $F(k, X(k), u(k)) \in \mathbb{R}^{n+1}$ is the state transition function. It represents all the expressions on the right-hand side of the system of token buckets given by (2.2) and those of the multiplexor given by (2.5). We will denote the components of this vector $F(k, X(k), U(k))$ by $F_i(k, \rho(k), u(k))$, $i = 1, 2, \ldots, n+1$. Let $\mathbb{N}_0$ denote the set of all nonnegative integers and $\mathbb{N}_0^k$ denote the Cartesian product of k copies of $\mathbb{N}_0$. It is important to note that

$$F : \mathbb{N}_0 \times \mathbb{N}_0^{n+1} \times \mathbb{N}_0^{n} \longrightarrow \mathbb{N}_0^{n+1} \tag{2.8}$$

and consequently our system is governed by a vector difference equation in the state space $\mathbb{N}_0^{n+1}$.

3. Optimization of system performance

The basic objective of any computer communication network is to transfer data (represented in various formats) from point to point promptly, efficiently, and reliably without failure. Before we can perform any optimization, we must define an appropriate objective functional that is reflective of these attributes. In view of the basic objectives, it is important to include in the functional all the possible losses that may occur in the network. We have seen that losses occur at the TBs and also at the multiplexor. In addition to these losses, there are delays in service due to the waiting time at the multiplexor. All these factors are included in the following objective functional:

$$J(u) = \sum_{k=0}^{K} \alpha(t_k) L(t_k) + \sum_{k=0}^{K} \sum_{i=1}^{n} \beta_i(t_k) r_i(t_k) + \sum_{k=0}^{K} \gamma(t_k) q(t_k). \tag{3.1}$$

The first term represents the weighted cell or traffic losses at the multiplexor. The second term gives the sum of weighted losses at the TBs and the last term represents the weighted cost of waiting time. The parameters $\alpha(t_k)$, $\beta_i(t_k)$, $\gamma(t_k)$, $i = 1,2,\dots,n$ are the weights to which different values can be assigned according to different scenarios and concerns. Note that, the objective functional is a function of u, which is the control vector. Our goal is to find an optimal control u that minimizes the cost function (3.1). We may recall that by the control u we mean the token supply to each of the individual TBs during each of the intervals of time $[t_{k-1}, t_k)$ over the entire period of operation.

Optimal control theory, such as the principle of dynamic programming or the minimum principle of Pontryagin, can be used to minimize the cost functional. In this paper, we use the principle of dynamic programming, introduced by Bellman [2], combined with a genetic algorithm to solve the optimization problem. Because of the recursive nature of our system model, the problem can be broken up into simpler subproblems easily. In order to reduce the search time for control values in each subproblem of dynamic programming, we use genetic algorithm to find the global optima. By combining the power of both the dynamic programming and the genetic algorithm, we can determine the optimal control values effectively without using excessive iterations involving all the admissible control values. In fact genetic algorithm partially improves the performance of dynamic programming.

3.1. Principle of dynamic programming for discrete time optimization. Consider the system governed by the state equation with $X(k)$ denoting the state at time k and $u(k)$ the control:

$$\begin{gathered} X(k+1) \equiv F(k, X(k), u(k)), \quad k = 0,1,2,\dots,N-1, \\ X(0) = X_0. \end{gathered} \tag{3.2}$$

The controls are constrained to belong to a specified set $\mathcal{U}_{\text{ad}}$ called the class of admissible controls. The cost functional is given by

$$J(u) \equiv \sum_{k=0}^{N-1} \ell(k, X(k), u(k)) + W(N, X(N)), \tag{3.3}$$

where the first term represents the running cost and the last term determines the terminal cost. The objective is to find a control policy from the admissible class $\mathcal{U}_{\text{ad}}$ that minimizes the functional J. We formulate this as a dynamic programming problem. Given the state $X(r)$ at time r, we may define

$$J(r,X(r),u) \equiv \sum_{k=r}^{N-1} \ell(k,X(k),u(k)) + W(N,X(N)) \tag{3.4}$$

as the cost of operating the system from time r starting from state $X(r)$ and using the control policy $u \in \mathcal{U}_{\text{ad}}$.

Now we may define

$$V(r,X(r)) \equiv \inf\{J(r,X(r),u),\ u \in \mathcal{U}_{\text{ad}}\} \tag{3.5}$$

as the value function. Clearly this gives the minimal cost to run the system starting from state $X(r)$ at time r till the end of the period N. Using the basic arguments of dynamic programming one can readily prove that the value function V must satisfy the following recursive equation:

$$\begin{gathered} V(k,x(k)) \equiv \operatorname*{Inf}_{u\in\mathcal{U}_{\text{ad}}} \{\ell(k,x(k),u(k)) + V(k+1,F(k,x(k),u(k))),\ k = 0,\dots,N-1\}, \\ V(N,x(N)) \equiv W(N,x(N)). \end{gathered} \tag{3.6}$$

This is known as the Bellman equation of dynamic programming. To solve this equation, we can use Lagrange multipliers [2, 9], or most easily a quadratic programming algorithm [2, 9]. However, since our system evolves in the lattice $\mathbb{N}_0^{n+1}$, we have to consider integer constraints. Some algorithms have been used, such as iterative dynamic programming (IDP) [8]. In our work, due to the high dimension of the problem [2, 9], the state and control sets are spread over a very large range of multiple integers. Thus we consider the use of genetic algorithms [5] which should allow us to find the optimal solution without having to explore all the possible solutions.

3.2. Dynamic programming equation for token-bucket mechanism. Note that, in our system, there are n traffic sources that result in n states associated with token buckets and one associated with the multiplexor queue giving $n+1$ state variables. We use $X(k)$ to represent the state vector where k is the stage variable that keeps track of the stage index. The control $u(k)$ in dynamic programming is defined as the vector of token generation rates. Because n TBs are considered, each TB is controlled by its own token generation policy. Thus $u(k) \in \mathbb{N}_0^n$ is a n vector as well.

To solve the optimal control problem from time t_0 to t_N, we divide the problem into multiple stages indexed from 0 to N. Let $k = 0,1,\dots,N$ be the stage variable. Each stage takes one time interval $[t_k,t_{k+1})$. Therefore, stage $k = 0$ represents time interval $[t_0,t_1),\dots,$ $k = N-1$ represents $[t_{N-1},t_N)$, and $k = N$ represents t_N, respectively.

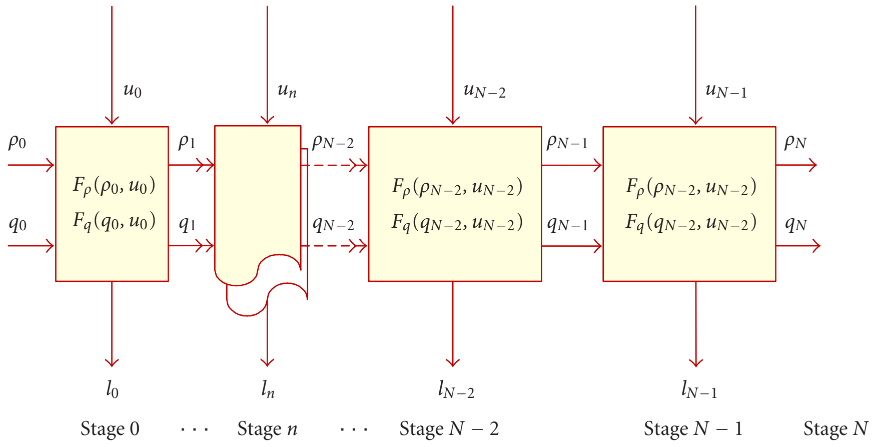


Figure 3.1. Multistage process of token-bucket control problem.

Our task is to study the high-dimensional integer optimization problem using the discrete time dynamic programming as discussed in the previous section. Accordingly, the complete set of equations required to solve the problem consists of the Bellman equation, the cost functional, and the state equation. These are all collected at one position for the convenience of the reader as follows:

$$V(k,X(k)) \equiv \inf_{u\in\mathcal{U}_{\text{ad}}} \{\ell(k) + V(k+1,F(k,X(k),u(k)))\}, \qquad V(N,X(N)) = 0, \tag{3.7}$$

$$\ell(k) = \alpha(k)L(k) + \sum_{i=1}^{n} \beta(k)r_i(k) + \gamma(k)q(k), \tag{3.8}$$

$$X(k) \equiv (\rho(k),q(k)), \qquad F \equiv (F_\rho,F_q), \qquad X(k+1) = F(k,X(k),u(k)), \tag{3.9}$$

$$\begin{aligned} F_\rho(k,\rho(k),u(k)) \equiv{}& \rho(t_k) + u(t_k) \wedge (T-\rho(t_k)) \\ &- v(t_k)I(v(t_k) \le [\rho(t_k) + u(t_k) \wedge (T-\rho(t_k))] \wedge c\tau), \end{aligned} \tag{3.10}$$

$$\begin{aligned} F_q(k,\rho(k),q(k),u(k)) \equiv{}& \{(q(t_k) - C\tau) \vee 0\} \\ &+ \left[\sum_{i=1}^{n} g_i(t_k)\right] \wedge [Q - (q(t_k) - C\tau) \vee 0]. \end{aligned} \tag{3.11}$$

3.3. Solution of the dynamic programming equation. By using the dynamic programming equation (DPE) (3.7) along with the terminal condition, we solve our optimal traffic control problem step by step. This is a sequential approach which eventually leads to the optimal control policy u^*. Before describing the computational sequence, we discuss how the optimal value can be obtained by dynamic programming. In our network traffic control system, the multistage process is shown in Figure 3.1 as follows.

The rectangle represents the transition function that contains two operations F_ρ and F_q where ρ and q denote the state variables indicating the states of the TBs and the multiplexor with u denoting the decision variable representing the token supply. As shown in Figure 3.1, the output of one stage is the input of the next one. To find the minimum cost (equivalently the value) during time $[t_0, t_N]$, we must find the optimal control value u^* which minimizes the cost of the whole process from stage 0 to stage N giving $V(0,X(0))$. It is clear from the DPE (3.7) that the optimum cost $V(0,X(0))$ is the sum of $\ell(0)$ and $V(1,X(1))$. Thus to compute $V(0,X(0))$ we must know $V(1,X(1))$ before we can find the optimal $u^*(0)$ that minimizes the cost to go from $X(0)$ giving $V(0,X(0))$. This is true for every stage. Hence the DPE must be solved backward in time starting from the last stage. At the terminal time N, $V(N,X(N)) = W(N,X(N))$ for a given function W. To minimize the terminal cost one must choose $u(N-1)$ that minimizes the functional $W(N,F(N-1,X(N-1),u(N-1)))$. Clearly the optimal $u(N-1)$ depends on the state at stage $N-1$ and so it may be denoted by $u(N-1,X(n-1))$. This is continued till the initial stage is reached. If the terminal cost is zero, this process begins from stage $N-1$. The whole process is clearly illustrated in Figures 3.2 and 3.3.

3.4. Reduction of the search time using a genetic algorithm. The high-dimensional nature of the token-bucket control system makes the number of combinations of states very large. The search space for the control values for each state at each stage is also large. For each TB, the admissible control values are $0,1,\dots,(T_i - \rho_i(t_k))$. Thus there are $(T_i - \rho_i(t_k) + 1)$ choices for the ith token bank at time t_k. If there are n TBs, the total number of admissible control values is given by

$$\prod_{i=1}^{n} \left(T_i - \rho_i(t_k) + 1\right). \tag{3.12}$$

For 3 TBs with the same maximum capacity 3000, the total number of admissible controls would be $[(1500-0)+1]^3 = 3\,381\,754\,501$. Obviously, to find an optimal value from such a large set is not easy. A genetic algorithm may be used to efficiently solve this problem.

Genetic algorithm works very well on combinatorial problems. They are less susceptible to falling into local extrema than other search methods such as gradient search, random search. This is due to the fact that the genetic algorithm traverses the search space using the genotype rather than the phenotype [17]. However, GAs tend to be computationally more intensive than the latter ones. A genetic algorithm is a search procedure that optimizes a certain objective function by maintaining a population of candidate solutions. It employs some operations that are inspired by genetics to generate a new population with better fitness from the previous one. A typical GA should consist of a population, chromosome, fitness, crossover operator, and mutation operator. For the detailed description of those components, the reader may refer to [5].

In our implementation, to avoid the loss of the best individuals and speed up the search process, we use the steady-state GA to solve the optimal traffic control problem. Different from simple GA, steady state GA only replaces a few individuals in each generation. The number or the percentage of replacement can be user specified. This type of replacement is often referred to as overlapping populations [17].

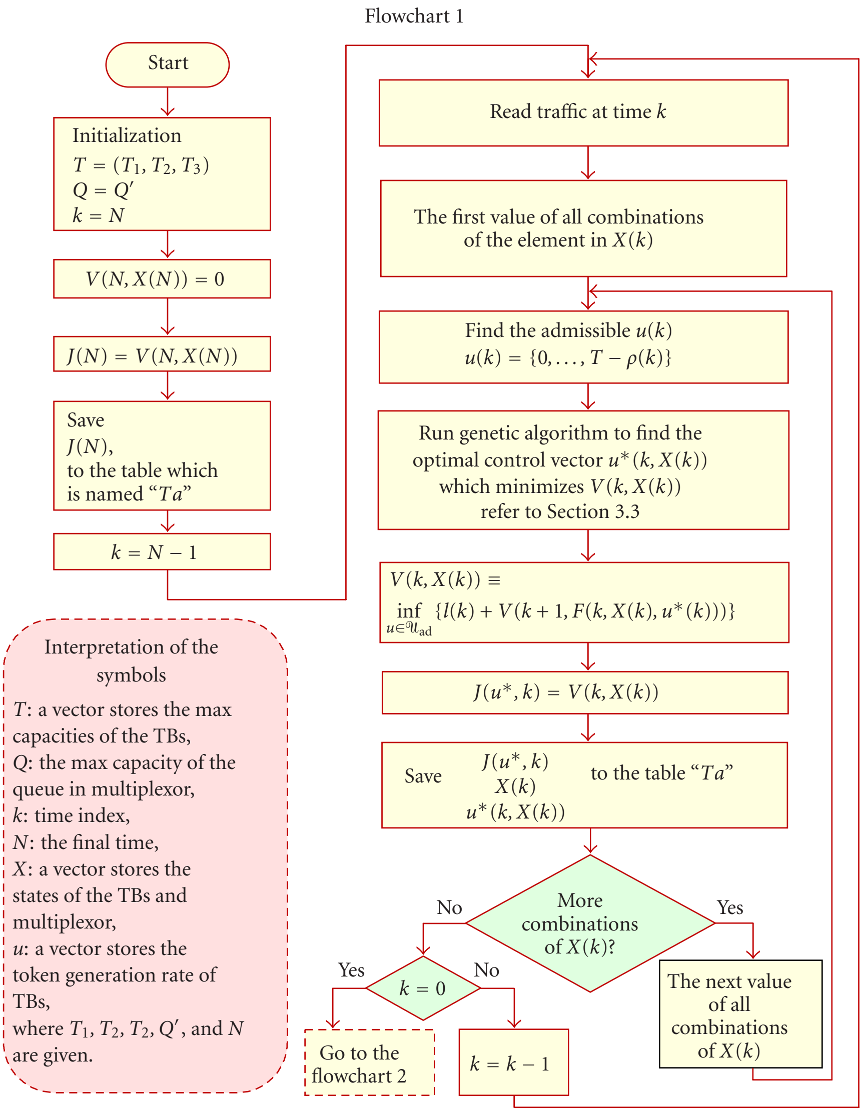

Flowchart 1
Start
Initialization
$T = (T_1, T_2, T_3)$
$Q = Q'$
$k = N$
$V(N, X(N)) = 0$
$J(N) = V(N, X(N))$
Save
$J(N)$,
to the table which
is named "Ta"
$k = N - 1$
Interpretation of the symbols
T: a vector stores the max capacities of the TBs,
Q: the max capacity of the queue in multiplexor,
k: time index,
N: the final time,
X: a vector stores the states of the TBs and multiplexor,
u: a vector stores the token generation rate of TBs,
where T_1, T_2, T_2, Q', and N are given.
Read traffic at time k
The first value of all combinations of the element in $X(k)$
Find the admissible $u(k)$
$u(k) = \{0, \dots, T - \rho(k)\}$
Run genetic algorithm to find the optimal control vector $u^*(k, X(k))$ which minimizes $V(k, X(k))$ refer to Section 3.3
$V(k, X(k)) \equiv \inf_{u \in \mathcal{U}_{ad}} \{l(k) + V(k+1, F(k, X(k), u^*(k)))\}$
$J(u^*, k) = V(k, X(k))$
Save $J(u^*, k)$ $X(k)$ $u^*(k, X(k))$ to the table "Ta"
More combinations of $X(k)$?
No
Yes
$k = 0$
Yes
No
Go to the flowchart 2
$k = k - 1$
The next value of all combinations of $X(k)$

Figure 3.2. Optimal control token-bucket algorithm flowchart (part 1).

For the control problem, the search space is the entire set of admissible control vectors u. A population is composed of a small group of selections from the whole control set. Each individual (chromosome) within a population represents a combination of each control value of each TB. If there are n TBs, each individual contains n genes, each gene maps one control value of one TB. The formation of chromosome and the formation of population are shown in Figure 3.4. The fitness is calculated by the value function.

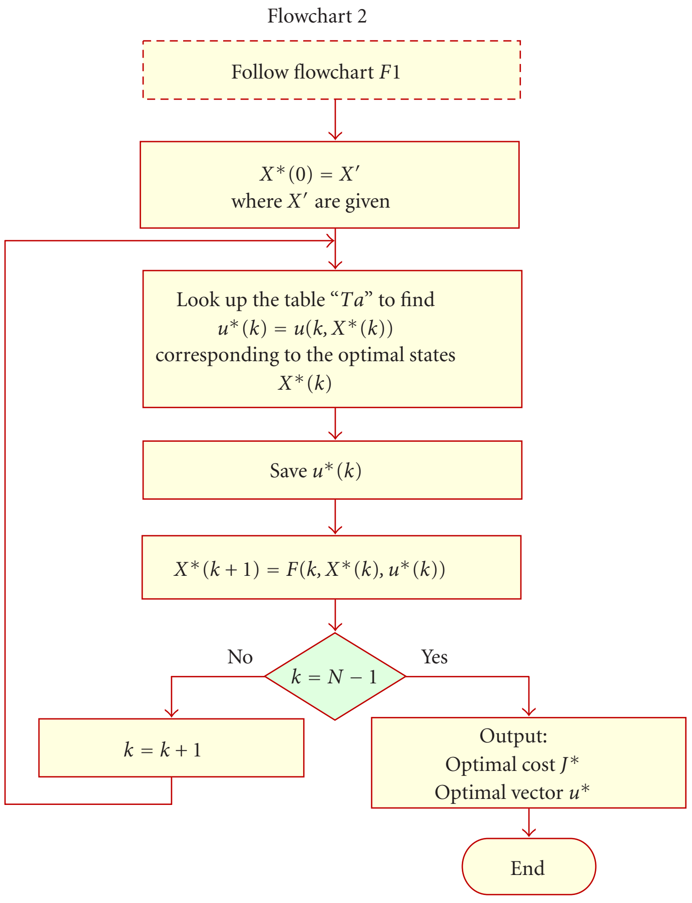

Flowchart 2
Follow flowchart $F1$
$X^*(0) = X'$ where X' are given
Look up the table "Ta" to find $u^*(k) = u(k, X^*(k))$ corresponding to the optimal states $X^*(k)$
Save $u^*(k)$
$X^*(k+1) = F(k, X^*(k), u^*(k))$
$k = N - 1$
No
Yes
$k = k + 1$
Output: Optimal cost J^* Optimal vector u^*
End

Figure 3.3. Optimal control token-bucket algorithm flowchart (part 2).

Before concluding this section, we present an outline of the steady-state genetic algorithm [13, 16].

(1) *Start*. Generate random population of M chromosomes (n vectors of admissible u).

(2) *Clone*. Copy the previous population to be the temporary population of the next generation.

(3) *Replace*. Replace some of the individuals by repeating the following steps until the user specified number of replacements or percentage of replacement is reached.

(3.1) *Select*. Select two parent chromosomes from the population according to their fitness (the better fitness, the bigger chance to be selected).

(3.2) *Crossover*. With a crossover probability cross over the parents to form a new offspring (children).

(3.3) *Mutate*. With a mutation probability, mutate new offspring at each locus (position in chromosome).

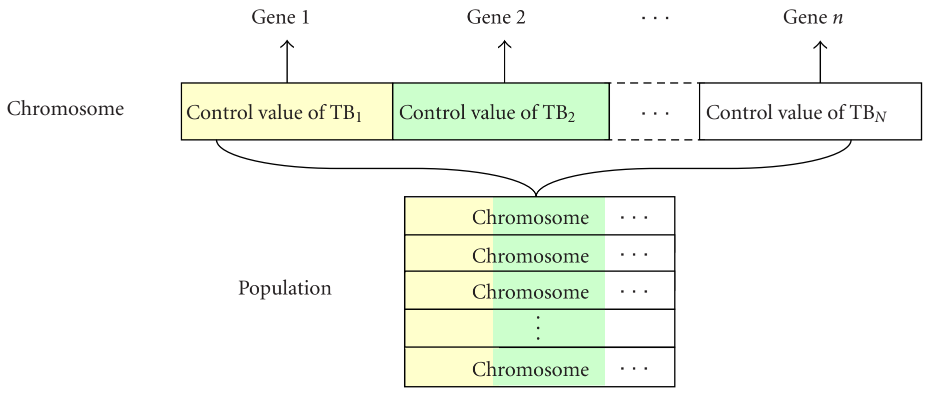


Figure 3.4. Formation of genome and population.

(3.4) *Return*. Place new offspring into the temporary population in the new generation.

(3.5) *Destroy*. Destroy the worst chromosomes to keep the specified population size.

(4) *Generate new population*. Use newly generated population for a further run of algorithm.

(5) *Test*. If the end condition is reached, *stop*, and return the best solution from the current population, else go to step 2.

The above algorithm has been implemented using the C++ programming language. We have made use of the Galib genetic algorithm package [10], written by Matthew Wall at the Massachusetts Institute of Technology. The source code was downloaded from [10].

For illustration, we consider two traffic sources. The traffic from each source provides the bit rates statistics (video frame sizes) of a traffic trace. After running the program, a set of optimal control strategies u^*, which represents the number of tokens generated during each stage, is obtained and stored.

4. Numerical results

In this section, we undertake the experimental evaluation of the algorithm previously introduced. Because the dynamic programming is computationally very intensive, we consider a scenario comprised by only two traffic sources policed by 2 token buckets.

4.1. Definitions of some measures of network efficiency. Before describing our numerical experiments and their analysis, we define the main performance metrics of interest. These metrics are key in the analysis of the network performance. We define the network utilization η, as follows:

$$\eta = \frac{\sum_{i=1}^{n}\sum_{k=0}^{K} v_i(t_k) - \left[\sum_{k=0}^{K} L(t_k) + \sum_{i=1}^{n}\sum_{k=0}^{K} r_i(t_k)\right]}{C(t_K - t_0)}. \tag{4.1}$$

The network average throughput is given by

$$\begin{aligned}\Upsilon = C\eta &= \frac{\sum_{i=1}^{n}\sum_{k=0}^{K} v_i(t_k) - \left[\sum_{k=0}^{K} L(t_k) + \sum_{i=1}^{n}\sum_{k=0}^{K} r_i(t_k)\right]}{(t_K - t_0)} \\ &= \frac{1}{t_K - t_0}\sum_{k=0}^{K}\Upsilon(t_k) \\ &= \frac{1}{t_K - t_0}\sum_{k=0}^{K}\left\{\left[\sum_{i=1}^{n} g_i(t_k)\right] \wedge \left[Q - \left(\left[q(t_k) - C\tau\right] \vee 0\right)\right]\right\},\end{aligned} \tag{4.2}$$

where $\Upsilon(t_k)$ denotes the throughput during the time interval $[t_{k-1}, t_k)$.

4.2. Specification of the traffic traces and system setup. Multimedia services tend to be highly demanding in terms of network resources. In the near future, a major part of the traffic will be produced by multimedia services, such as voice and video. In particular, traffic flows generated by video applications have the properties of high peak rate and strong variability: two features presenting serious challenges to the provisioning of QoS guarantees.

The traces used in our experiments correspond to the traffic statistics of two MPEG-1 sequences encoded using the Berkeley MPEG-encoder (version 1.3). Each group of pictures (GoP) contains 12 frames with a frame rate of 24 frames/s. Traces 1 and 2 represent the traffic trace of 7 seconds of the movies "Mr. Bean," and "The Simpsons," respectively. Both video traces have been encoded according to the video frame pattern *IBBPBBPBBPBB*. According to the MPEG-1 encoding scheme, the I-frames are encoded with a moderate compression ratio, P-frames have a higher compression ratio than I-frames, and B-frames have the highest compression ratio [3]. The traffic traces for both video streams are shown in Figure 4.1.

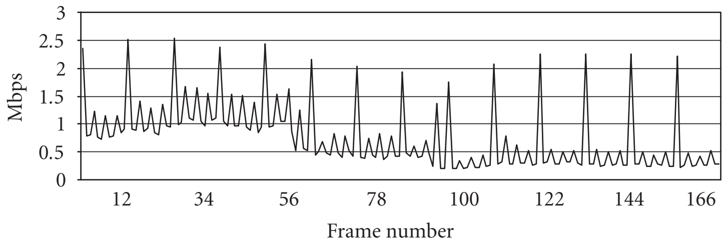
From the figure the GoP pattern is clearly identified. An I-frame appears at a regular interval of one every 12 frames. The video traffic statistics are given in Table 4.1.

For the transmission of the video data through the network, we assume the use of the UDP/IP protocol stack. Each frame is divided into one or more packets with varying sizes between 64 bytes to 1500 bytes. Without loss of generality, we further consider that one token takes on 100 bytes of traffic.

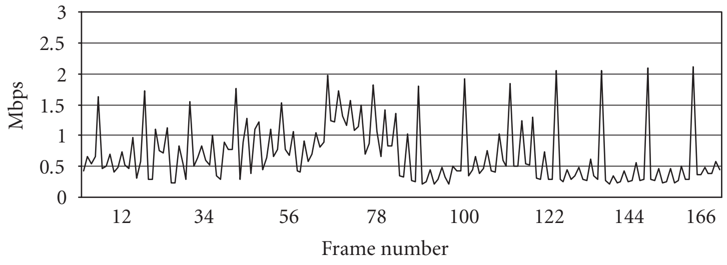
Throughout the experiment, relative weights are assigned to the different losses as follows:

(i) $\alpha(t_i) = 10$, weight given to the multiplexor losses;
(ii) $\beta(t_i) = 5$, weight assigned to the losses at the TB;
(iii) $\gamma(t_i) = 1$, weight given to the waiting losses.

Under the scenario considered herein, by assigning more weight to the losses at the multiplexor, we are considering it preferable to reject the traffic at the TB (the entry point to the network) than to have to drop the packets in excess at the multiplexor. This is in line with the IETF philosophy that once the (conforming) traffic has been admitted into the network, enough resources have to be guaranteed for its transmission. We also consider the tradeoff between the losses at the TBs and the waiting losses. If the multiplexor has enough space to accept more traffic, the traffic may be accepted even though it may



(a)



(b)

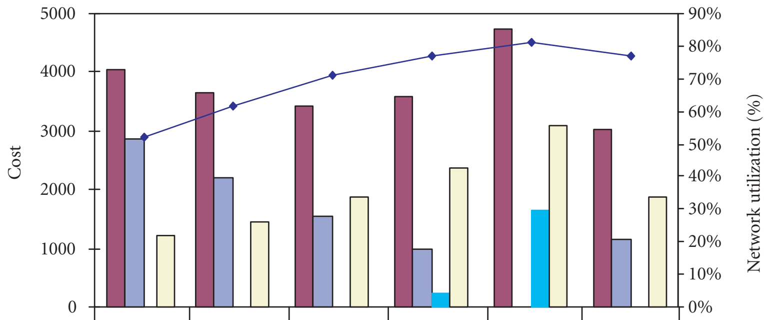
Figure 4.1. MPEG-1 traffic traces (a) 1 and (b) 2.

Table 4.1. Summary of MPEG-1 traffic traces specification.

Traffic trace	Peak rate (Pi)	Average rate (Mi)	Standard deviation (Si)
Trace 1	2.54 Mbps	0.759 Mbps	0.5784 Mbps
Trace 2	2.104 Mbps	0.696 Mbps	0.477 Mbps

experience at times longer delays. Therefore, the weight assigned to the losses at the TB is larger than the weight given to the delay caused by the waiting losses. It is understood that depending on the application and policy implemented in a particular setup, the relative weights will have to be set up accordingly.

Results analysis. We have conducted a set of experiments to observe the dependence of the cost function on different control strategies. The total cost computed by (3.1), total losses at the TBs, losses at the multiplexer, and the waiting losses, corresponding to the optimal control, will be compared with those corresponding to some other open loop controls with fixed token generation rates. More importantly, traces of the conforming traffic, bucket occupancy, and token generation rates will allow us to understand the operation of the token-bucket mechanism. By studying these traces, we should be able to



	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5	Strategy 6
🟥 $J(u)$	40 592	36 520	34 058	35 855	47 360	30 099
🟪 TB losses	28 570	21 975	15 465	9885	0	11 360
🟦 Multiplexor losses	0	0	0	2230	16 330	0
🟨 Waiting losses	12 022	14 545	18 593	23 740	31 030	18 739
—◆— Network utilization (%)	52.13%	61.55%	70.85%	77.23%	81.28%	76.72%

Figure 4.2. Dependence of cost on control strategy ($C = 1.456$ Mbps).

gain insight in the mode of operation of all the control strategies under consideration. The system configurations are specified as follows.

(i) $T_i = 2000$ bytes, $i = 1, 2$: the token bucket has been set large enough to be able to accommodate the maximum packet length.

(ii) $C = \sum_{i=1}^{2} M_i = 1.456$ and 2.184 Mbps: these two values have been chosen to evaluate a system configuration overloaded or heavily loaded.

(iii) $c_i = 2.56$ Mbps: the rate of the link joining the TB to the multiplexor is assumed large enough to deliver the conforming traffic at the rate it is being generated.

(iv) $Q = 4000$ bytes: the multiplexor has been dimensioned to accommodate up to two 1500-byte packets.

(v) $\tau = 4.63$ milliseconds: this interval corresponds to the arrival of a packet, since frames are generated at a rate of 24 frames per second, we assume that a video frame is segmented into up to nine packets.

(vi) $\rho_i(0) = T_i$: the token buckets are assumed to be initially full.

The following cases are studied.

(1) Open loop controls without optimization.

(i) Strategy 1: $u_i(t_k) = 0.7 \times M_i$.

(ii) Strategy 2: $u_i(t_k) = M_i$.

(iii) Strategy 3: $u_i(t_k) = 1.3 \times M_i$.

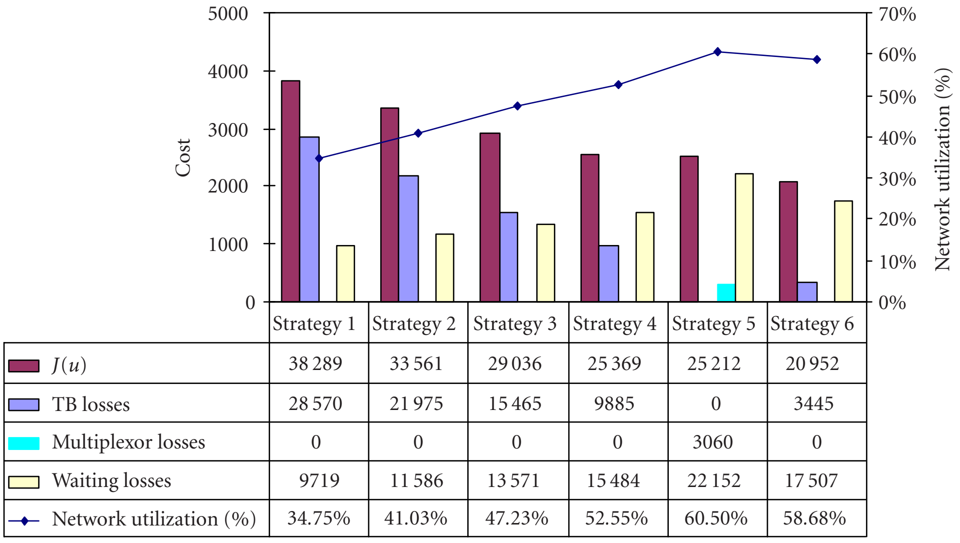
(iv) Strategy 4: $u_i(t_k) = 1.6 \times M_i$.

(v) Strategy 5: $u_i(t_k) = P_i$. (peak rate of the ith traffic).

(2) Optimal control.

(vi) Strategy 6: with variable bit rate control.

Numerical results are shown in Figures 4.2, 4.3, and 4.4.

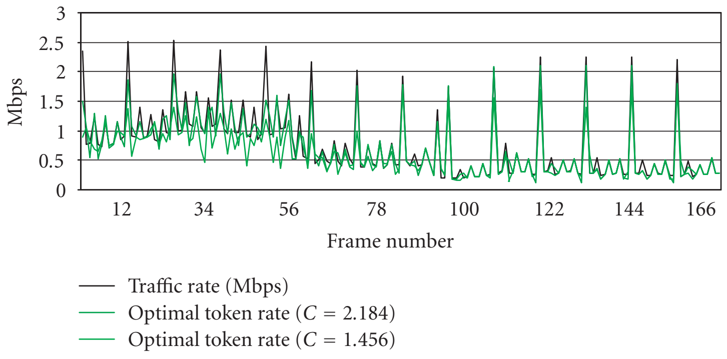


	Strategy 1	Strategy 2	Strategy 3	Strategy 4	Strategy 5	Strategy 6
🟥 $J(u)$	38 289	33 561	29 036	25 369	25 212	20 952
🟪 TB losses	28 570	21 975	15 465	9885	0	3445
🟦 Multiplexor losses	0	0	0	0	3060	0
🟨 Waiting losses	9719	11 586	13 571	15 484	22 152	17 507
◆ Network utilization (%)	34.75%	41.03%	47.23%	52.55%	60.50%	58.68%

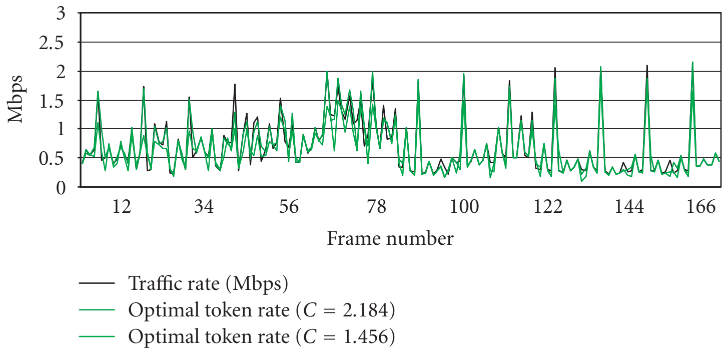
Figure 4.3. Dependence of cost on control strategy ($C = 2.184$ Mbps).

Figures 4.2 and 4.3 show that despite the output link capacity increase, strategies 1, 2, and 3 exhibit the same performance, that is, the TB losses cannot be reduced by simply increasing the output link capacity, C. In the case of $C = 1.456$ Mbps, strategy 5, which corresponds to the open loop control with a fixed token generation rate equal to the traffic's peak rate, provides the worst performance of all. Even in the case when the channel rate is increased to $C = 2.184$ Mbps, this strategy is so aggressive that still some packets are lost at the multiplexor. In applications, such as video communications, losses may result in low quality of the video at the receiving end or more delay due to lost packet retransmission. Long waiting time of the packet at the multiplexor may also cause unacceptable delay of video frames. In some applications, such as packetized voice, long delays may considerably affect the quality of service of the end application. In this case, it may be better to drop the packets in excess. It is clear that strategies 1 and 2 are too conservative resulting in a high number of packets being dropped at the TB. By optimizing the operation of the overall system, strategy 6 balances the packet losses while ensuring a proper level of utilization.

Figure 4.4 depicts the optimal token generation rates and traffic rates for both TBs. We observe that the token generation rates follow the same pattern as the traffic traces. In situations when both sources exhibit higher rates at the same time, the token generation rates are limited by the output channel rate. For instance, from the figure we see that at the large spike around frame 50, the rates of both sources add up to about 4 Mbps. However, the token generation of both sources are clearly limited by the channel output rate. For both channel rates, the token generation rate of source 1 is lower than the corresponding traffic rate. This will obviously translate into packets that will have to be discarded at the token bucket.



(a)

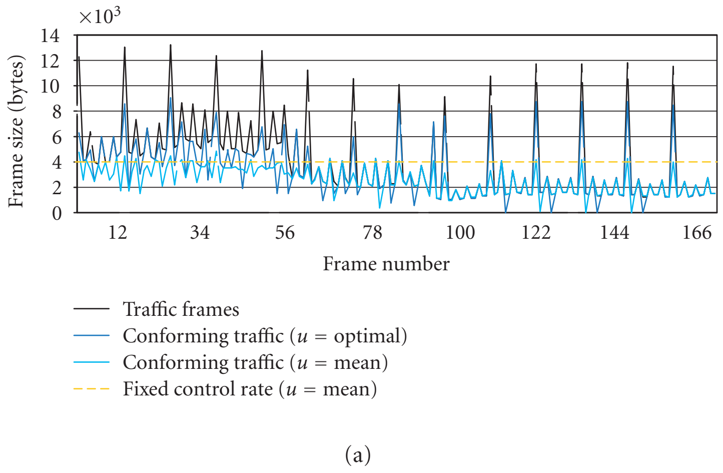


(b)

Figure 4.4. Traffic rate versus optimal token generation rates: (a) TB_1 and (b) TB_2.

Figure 4.4 also shows that in the case when one of the sources transmits at a much higher rate than the other one, the token generation rate of the high activity source will follow accordingly. In the mean time, the token generation rate of the low activity source may not be able to generate the required tokens to transmit all its packets. For instance, at frame 109, the traffic rate of source 1 exceeds the 2 Mbps while source 2 is transmitting at a rate of about 0.4 Mbps. As seen from the figures, the token generation rate of source 1 is slightly higher than 2 Mbps, while the token generation rate of source 2 shows a significant discrepancy with respect to the source rate.

Figure 4.5 provides the traces of the incoming traffic, together with the conforming traffic resulting when applying strategies 2 (token generation rate = mean) and 6 (the optimal strategy). For reference purposes, the figures also show the mean rate of the input traffic. As expected, it is clear that by fixing the token generation rate equal to the mean rate of the sources, there is considerable waste of resources. In the case of the optimal



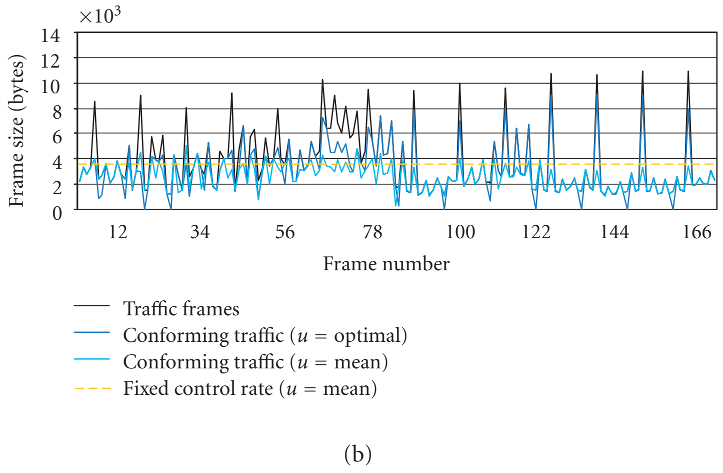
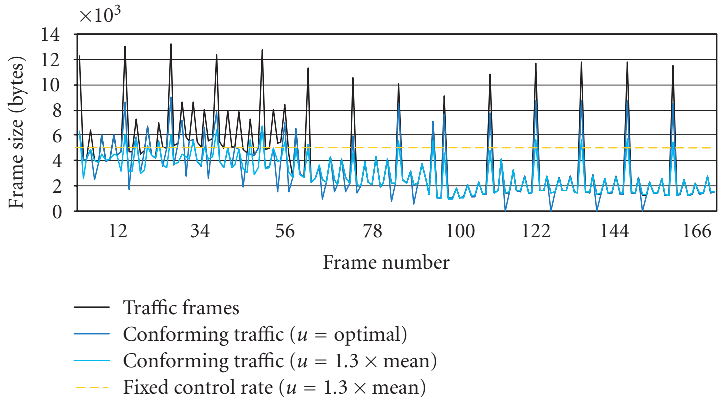
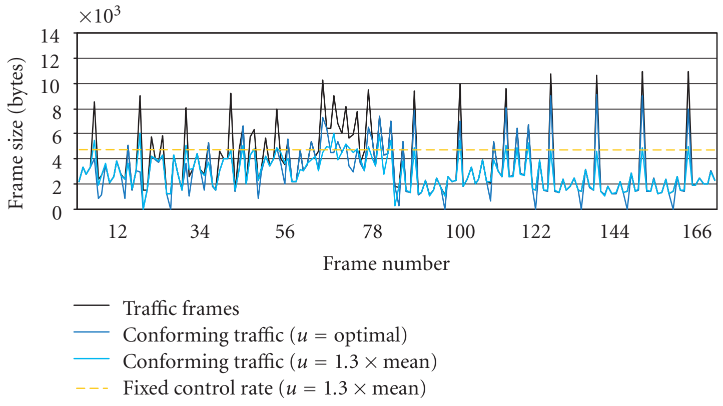


Figure 4.5. Incoming traffic versus conforming traffics (a) TB_1 and (b) TB_2 (fixed u = mean traffic).

strategy, the resources are dynamically distributed. We notice however that at some points of time, the rate of the conforming traffic exhibits very low values. This happens at time instants when one source transmits at a much higher rate than the other one. For instance, at frame 109, there is a considerable discrepancy between the actual incoming traffic rate and the rate of the conforming traffic. This results from the fact that the optimization has been done for the overall system as a whole. From Figure 4.4 we have seen that the token generation rate of source 2 exhibits this same discrepancy with respect to the input traffic rate. Once again, a satisfactory solution to this unfair condition will depend on the application or policy in place. For instance, for the results from the particular setup



(a)

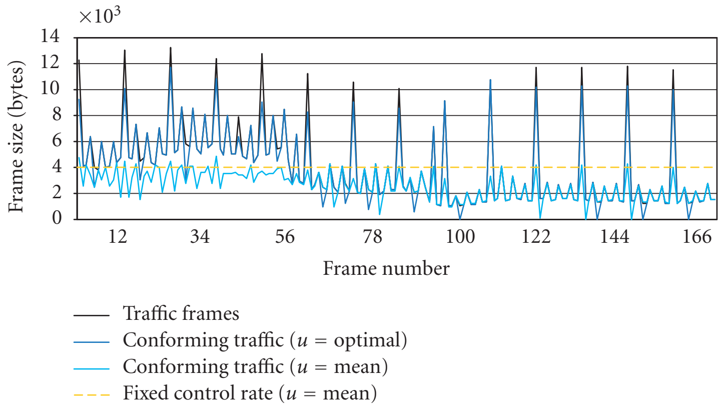


(b)

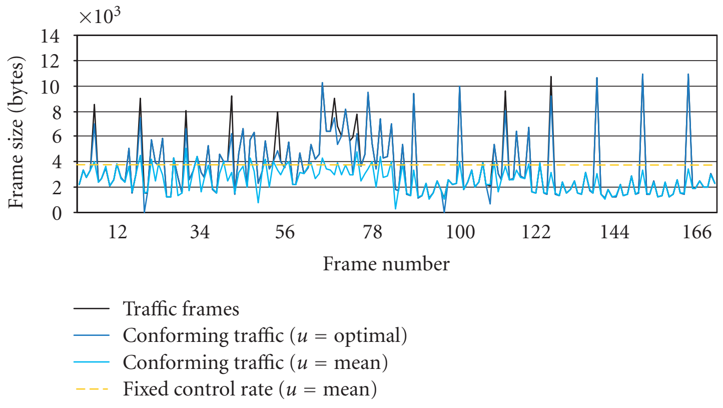
Figure 4.6. Incoming traffic versus conforming traffics (a) TB_1 and (b) TB_2 (fixed $u = 1.3\times$ mean traffic).

considered herein, we see that the sources take turns in gaining a bigger share of the resources, that is, channel bandwidth.

From Figure 4.5 we see that there are situations when the conforming traffic gets shut down. For instance, at frame 121, the conforming traffic of source 2 is completely shut down, and this is true despite that there are tokens being generated during this frame at its corresponding token bucket (refer to TB_2 in Figure 4.4). However, since the packet length being used can be as large as 1500 bytes, there may not be enough tokens present at the token bucket to take on the whole packet. According to our model, in this case, the whole packet is discarded at the token bucket. In the figure, there are several instances



(a)

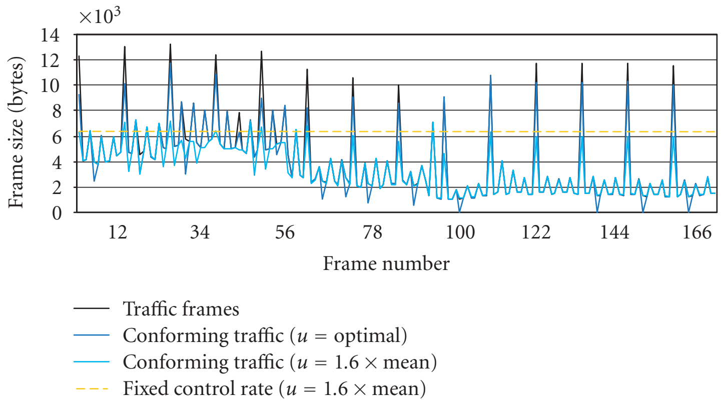


(b)

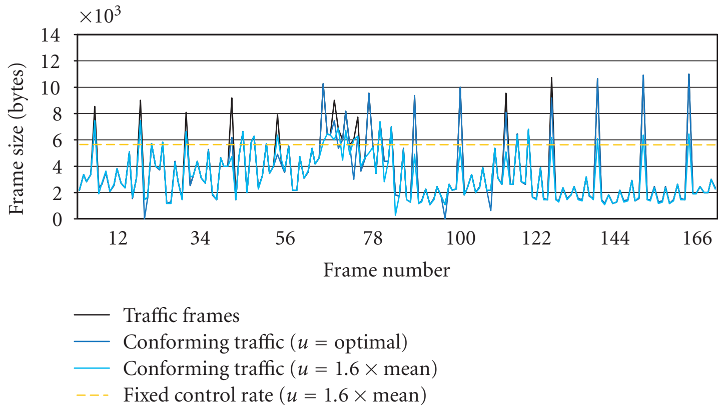
Figure 4.7. Incoming traffic versus conforming traffics (a) TB_1 and (b) TB_2 (fixed u = mean traffic).

where this situation arises for both sources. Figure 4.6 shows similar results as Figure 4.5 except that in this case, the token generation rate has been fixed to 1.3 times the mean of the input traffic rate. From the figure, it is clear that setting the token generation rate at a higher rate may help during period where the rate of a source remains high during a long period of time. For instance, during the frames 64 to 72, source 2 exhibits a high degree of activity. In the case when the token generation rate is set to 1.3 times the mean, the conforming traffic exhibits at some points even higher values than the ones obtained for the optimal case.

Figures 4.7 and 4.8 show the traces of the conforming traffic and sources for the case when the capacity of the link, $C = 2.184$ Mbps. In Figure 4.7 there are still some instances



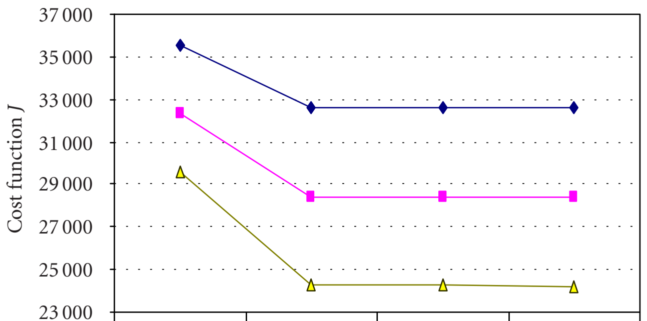
(a)



(b)

Figure 4.8. Incoming traffic versus conforming traffics (a) TB_1 and (b) TB_2 (fixed $u = 1.3\times$ mean traffic).

when the conforming traffic is completely shut down. However, overall, strategy 6 is able to take advantage of the extra capacity reducing considerably the overall losses while maintaining a good level of utilization. Furthermore, during period of high activity, it is able to make use of the extra capacity. For instance, during the frames 64 to 72, the conforming traffic corresponding to source 2 follows very closely the pattern of the incoming traffic. In the case of fixed token generation, even a token generation rate of up to 1.6 times the mean incoming traffic rate is not able to cope with the demand during this period of time as effectively as the optimal case (see Figure 4.8).



Buffer capacity (Q) - - ->	2500 bytes	4000 bytes	4500 bytes	5500 bytes
◆ $C = 1.456$ Mbps	35 500	32 590	32 590	32 599
■ $C = 1.758$ Mbps	32 337	28 360	28 360	28 360
△ $C = 2.184$ Mbps	29 569	24 243	24 237	24 233

Figure 4.9. Dependence of cost J on Q and C.

We now study the effect of some of the network parameters over the cost functions for the optimal control. In this set of experiments, we change the values of the buffer size (queue capacity) and the values of the output link service rate C, the values of the parameters and the corresponding costs are summarized in Figure 4.9. In Figure 4.9, M_i is the incoming traffic mean bit rate of ith source, where $i = 1, 2$. All the other parameters remain the same as in the previous set of experiments.

From Figure 4.9, as expected, increasing the output capacity C decreases the cost. However, increasing the buffer size Q beyond a certain size does not help to improve the system performance. Except for the case when the buffer size is set to be 2500 bytes, all other buffer sizes show similar cost values for a given output capacity rate. This is not surprising if we realize that the token-bucket size has been set to 2000 bytes, allowing the sources to transmit up to 2000 bytes back to back. Therefore a minimum buffer size of 4000 has to be provided to accommodate the conforming traffic. Furthermore, it is important to point out that the waiting losses do not play a major role as the multiplexor buffer is increased. This is due to the fact that the amount of conforming traffic present in the multiplexor at a given time will be limited by size of the token buckets.

Our results provide us with the basic guidelines towards the design of adaptive control mechanism capable of better distributing the network resources and fulfilling the needs of a wide range of applications.

5. Conclusion

The optimization of the token-bucket control mechanism we discussed in this paper allows us to gain insight into the operation of the token-bucket algorithm. The numerical results and the system analysis demonstrate the effectiveness of dynamic programming, genetic algorithm, and our system model in the study of the operation and performance of the token bucket. They also set the guidelines for the design of the system parameters, tradeoff over losses, delay and network utilization. Our future work will focus on the following points: (1) enhance the dynamic model to take into account issues, such as fairness, particular application requirements, and packet dropping policies; (2) design of an optimal feedback control strategy based on the lessons learnt herein; (3) improve the computation and memory requirements of the optimization algorithm presented in this paper. In order to further reduce the running time, we will be considering the use of parallel computing. Memory requirements is another issue that will have to be addressed. So far, we have not found a method to reduce the memory allocation. We may explore the "check point" theory [3, 14].

Acknowledgment

This work was partially supported by the National Science and Engineering Research Council of Canada under Grant A7109.

References

[1] N. U. Ahmed, Q. Wang, and L. Orozco Barbosa, *Systems approach to modeling the token bucket algorithm in computer networks*, Math. Probl. Eng. **8** (2002), no. 3, 265–279.
[2] R. Bellman, *Dynamic Programming*, Princeton University Press, New Jersey, 1957.
[3] C. Bruni and C. Scoglio, *An optimal rate control algorithm for guaranteed services in broadband networks*, Comput. Networks **37** (2001), no. 3-4, 331–344.
[4] R. Bruno, R. G. Garroppo, and S. Giordano, *Estimation of token bucket parameters of VoIP traffic*, Proc. IEEE Conference on High Performance Switching and Routing, Heidelberg, 2000, pp. 353–356.
[5] D. A. Coley, *An Introduction to Genetic Algorithms for Scientists and Engineers*, World Scientific, New Jersey, 1999.
[6] J. A. Grice, R. Hughey, and D. Speck, *Reduced space sequence alignment*, Comput. Appl. Biosci. **13** (1997), no. 1, 45–53.
[7] A. Lombaedo, G. Schembra, and G. Morabito, *Traffic specifications for the transmission of stored MPEG video on the Internet*, IEEE Trans. Multimedia **3** (2001), no. 1, 5–17.
[8] R. Luus, *Iterative dynamic programming: from curiosity to a practical optimization procedure*, Control and Intell. Syst. **26** (1998), no. 1, 1–8.
[9] G. L. Nemhauser, *Introduction to Dynamic Programming*, John Wiley & Sons, New York, 1966.
[10] M. Obitko, *Introduction to Genetic Algorithms*, 1998, http://cs.felk.cvut.cz/~xobitko/ga/.
[11] G. Procissi, A. Garg, M. Gerla, and M. Y. Sanadidi, *Token bucket characterization of long-range dependent traffic*, Comput. Commun. **25** (2002), no. 11-12, 1009–1017.
[12] A. S. Tanenbaum, *Computer Networks*, 3rd ed., Prentice-Hall PTR, New Jersey, 1996.
[13] P. P. Tang and T.-Y. C. Tai, *Network traffic characterization using token bucket model*, Proc. 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99), vol. 1, New York, 1999, pp. 51–62.
[14] C. Tarnas and R. Hughey, *Reduced space hidden Markov model training*, Bioinformatics **14** (1998), no. 5, 401–406.
[15] C. Wahida and N. U. Ahmed, *Congestion control using dynamic routing and flow control*, Stochastic Anal. Appl. **10** (1992), no. 2, 123–142.
[16] M. Wall, *A C++ Library of Genetic Algorithm Components*, http://lancet.mit.edu/ga/.
[17] ———, *Introduction to Genetic Algorithms*, http://lancet.mit.edu/~mbwall/presentations/IntroToGAs/.

[18] N. Yin and M. G. Hluchyj, *Analysis of the leaky bucket algorithm for on-off data sources*, Global Telecommunications Conference (Globecom '91), vol. 1, Arizona, 1991, pp. 254–260.

N. U. Ahmed: School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, Canada K1N 6N5
E-mail address: ahmed@site.uottawa.ca

Bo Li: School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, Canada K1N 6N5
E-mail address: lydia-lee8@hotmail.com

Luis Orozco-Barbosa: School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, Canada K1N 6N5
E-mail address: lorozco@info-ab.uclm.es