Network Traffic Characterization Using Token Bucket Model

Puqi Perry Tang and Tsung-Yuan Charles Tai
Intel Architecture Labs
Intel Corporation
2111 NE 25th Avenue
Hillsboro, OR 97124-5961
{Perry.Tang, Charlie.Tai}@intel.com

***Abstract* — This paper investigated the problem of deriving token bucket parameters from the observed traffic patterns of computer network flows in two cases. In the basic case, we identified the set of token bucket parameters for any given flow so that all the packets in the flow will be delivered immediately without incurring any delay or loss. Then, we extended the basic case by adding a queue to the token bucket model to temporarily store packets that cannot be delivered right away until enough tokens get accumulated in the bucket. The queue has the effect of smoothing the traffic so that the resulting token bucket parameters are less demanding. It also makes the derived token bucket parameters less fluctuated over time. Relationship among the queue size and token bucket parameters for a given flow are analyzed rigorously. Queuing delay for each packet in the flow is calculated and used to describe the adjusted post-queuing traffic pattern. Simple and efficient algorithms for the derivation of measurement-based traffic specification (MBTS) are presented, along with empirical results. This MBTS technology alleviates the need for users to explicitly characterize the traffic a priori in order to reserve network resource in an integrated services network.**

I. Introduction

A. Conventions and problems

Computer networks of integrated services, which provide multiple-level and dynamically selectable qualities of service (QoS) for different classes of traffic, rely on sophisticated network resource reservation and management technologies. To fulfill the service guarantee in an integrated services network [3], network traffic needs to be characterized a priori in terms of some deterministic or stochastic traffic models. This traffic characterization provides a bounding envelope on the traffic pattern that can be injected by the source to the network, and needs to be communicated to the network for resource reservation and admission control purposes. This paper focuses on the token bucket model (see [13, 9]) as adopted by the Internet Engineering Task Force (IETF) Integrated Services (IntServ) working group. One of the main challenges for such traffic characterization is to derive token bucket parameters of a network flow from its observed traffic pattern.

To be more precise, let's first introduce the notions of traffic pattern of a flow and the token bucket model [10]. Suppose a flow exists in a time interval $[t_0, t_n]$ and consists of n packets $p_1, p_2, ..., p_n$. Its traffic pattern $P = \{(p_i, t_i) \,|\, 1 \leq i \leq n\}$, where $t_0 < t_1 < t_2 < ... < t_n$, indicates that, at time $t_i \in [t_0, t_n]$, p_i is generated by an application in a host. Note that p_i is also used without ambiguity to denote the number of bytes of the i-th packet. A token bucket is a non-negative counter which accumulates tokens at a constant rate r until the counter reaches its capacity b. Upon arrival, a packet will be sent out with the token counter decreased by the packet size in bytes provided there are enough tokens for the packet. Otherwise, the packet will be either dropped, or queued until the token bucket accumulates enough tokens to send the packet out. When the flow is idle or running at a lower rate such that the token counter reaches the upper bound b, the accumulation of the token will be suspended till the arrival of next packet. Note that a token bucket is not a physical container of packets, but a numerical counter of tokens to schedule the releasing time of packets. A token bucket is parameterized by its token replenishment rate r, and the token bucket size b.

A key question for traffic characterization is that for a flow of pattern P in the time interval $[t_0, t_n]$, what would be the appropriate token bucket parameters (r, b) such that all packets could be sent out immediately upon their arrivals. In this case, we say the flow is compliant with the token bucket model parameterized by (r, b).

A token bucket model can be extended by adding a queue to it. This queue is used to hold the stream of packets while they are waiting for enough tokens to be accumulated in the bucket. With queuing, bursty traffic could be smoothed out so that the derived token bucket parameters become less demanding than those of the no-queue case, which results in more efficient resource allocation in the network. On the other hand, by adding a queue, a given token bucket model will be able to accommodate more varying traffic patterns at the expense of memory space and packet delay at the source.

There are several interesting questions to ask in the case of a token bucket with a queue. First of all, for a given flow and a token bucket model, what the corresponding minimal queue size q should be such that the flow does not suffer from any packet loss. Inversely, if the queue size q and one of r and b are given, what is the optimal choice for the other parameter? Moreover, what is the queuing delay, if any, for each of the packets in the queuing case? And if the maximum queuing delay is specified, how should we determine the minimal queue size and corresponding token bucket parameters?

B. Solutions of the problems

This paper gave explicit answers to all of the aforementioned questions with rigorous mathematical proofs and efficient algorithms, which are organized into two parts. The first part deals with the no-queue token bucket model, which is discussed in section II with three major results. First we found the lower and upper bounds for both b and r so that all the legitimate (r, b) pairs that satisfy the no-loss requirement fall in these ranges. Second, for any given r in its valid range, we can determine the optimal, i.e., the minimal b to couple with r so that no packet is dropped. This optimal b is expressed explicitly as an function of r, which is monotonously decreasing and piecewise linear. Third, we presented two efficient algorithms to compute this function and associated empirical results based on our implementations.

The second part, presented in section III, focuses on the token bucket model with queuing. We first identifies the bounds for valid r and b for a given queue size q. For a given traffic pattern, we found the minimal queue size q as a function of (r, b) pair so that no packet is dropped. This function $q(r, b)$ is decreasing in both r and b. On the other hand, for a given queue size q, we presented algorithms to compute the relations between optimal b and r. The latter is in fact a generalization of that for the no-queue case, i.e., these results are degenerated to those in section II if we set the queue size to zero.

Moreover, in section III, we explicitly calculated the queuing delay of each packet in the flow for a given token bucket model. This allowed us to derive the flow's maximal and average queuing delays, as well as the adjusted, post-queuing traffic pattern. Inversely, if an upper bound for the maximal delay or the average delay is specified for a flow, we presented algorithms to determine the set of token bucket models (r, b) that satisfy the delay bound. The computational complexity of these algorithms is analyzed. Empirical results illustrating the relationship among the queue size, token bucket parameters, and the delay bound are also presented.

C. Measurement-based traffic specification

In the current Internet QoS usage model, the sender of a flow is required to specify the flow characteristics a priori in order to allocate network resources [16, 12]. The most prominent way to describe a flow is to use the traffic specification (TSpec) as stipulated in the IETF Integrated Services specifications [13]. TSpec is comprised of parameters b and r of the token bucket model, the peak rate p, maximal packet size M and minimal policing unit m. This paper provides algorithms to determine the optimal b and r values, with the p and M values easily obtained as a byproduct in this process. The value of m needs to be separately configured.

With the results presented in this paper, Measurement-based Traffic Specification (MBTS) is made possible as an integral part of QoS services. As an example, RSVP-based QoS services [1] can take advantage of this technology by monitoring the traffic and generating TSpec on the fly either periodically or as needed. This dynamically generated TSpec reflects the most recent traffic pattern, and is used to update the existing network resource reservation. The MBTS technology has been implemented as an WinSock 2 layered service provider, and works transparently with the RSVP service in Windows environments [14]. This implementation was used to generate all the empirical results presented in this paper.

There is a tradeoff between transmission efficiency and bandwidth re-negotiation overhead. Higher transmission efficiency can be achieved by reducing the measurement period, which in turn detects and adjusts to traffic pattern changes quickly, but could introduce more frequent token bucket calculation and higher network protocol processing overhead for bandwidth re-negotiation. How to make the tradeoff in this aspect would be an important future research topic.

The MBTS technology enables the next generation QoS services, which have the following characteristics:

(1) Minimize the work required by the users of QoS services. Instead of explicitly specifying TSpec a priori, users merely need to indicate that a specific type of service is desired, and QoS services will measure the traffic and make appropriate reservation on behalf of the users automatically.

(2) Provide real-time tracking of traffic changes, precise measurement of traffic pattern, and adaptive bandwidth reservation. These capabilities remove the guesswork in the current QoS services, and maximize the network resource utilization.

In summary, the MBTS-enabled QoS services are automatic, precise, and adaptive.

D. Related work

The first, though unpublished, work to compute token bucket parameters from the traffic pattern was done by Craig Partridge and Mark Garrett in 1994 [11]. For a fixed r, they devised a single pass algorithm (called Send-Now) to derive the minimal value of b for the no-queuing case. They also devised another algorithm (called Send-Smooth) to accommodate relatively limited queuing case, in which a packet is only allowed to be delayed up to the point when the next packet arrives. Detailed description and comparison of these algorithms are presented in sections II.D and III.C, respectively.

S. Jamin, P.b. Danzig, S. Shenker and L. Zhang in [7] presented a specific algorithm and its simulation results for measurement-based admission control in support of predictive service, which allows for occasional delay violations and achieves higher network utilization. It relies on measuring the maximum delay and maximum bandwidth over a measurement interval to characterize existing flows in order to determine if a new flow can be admitted. However, a priori source characterization is still required for the to-be-admitted flow in this scheme, and our work in this paper is complementary.

M. Grossglauser and D. Tse in [6] studied the impact of estimation error of flow arrival and departure dynamics, and of measurement memory on the performance of the Measurement-based Admission Control system in terms of the overflow probability, which is a relatively simple service model. Their paper provides insights to the aforementioned issues by making simplifying assumptions, such as stochastic homogeneity across flows, stationarity, and the absence of long-range dependence(LRD) in the flows. These results, if extended to more realistic scenarios, could potentially further simplify the traffic descriptor requirement, but not totally eliminate that. Our work in this paper will simply remove the need for user to specific traffic characteristics a priori.

Researches have been done to evaluate the resource requirements (e.g., bandwidth and buffer size) in order to accommodate variable-bit-rate video streams for a given performance target (e.g., packet loss ratio) such as those done by M. Garrett and W. Willinger [4]. However, Garrett's work is based on the FIFO queuing with trace-driven simulation and statistical models from a single pre-recorded video session as opposed to the deterministic token bucket model and the mathematical analysis which can be efficiently applied to any traffic sessions on the fly as we have done in this paper. Also, Garrett's results are more applicable for resource allocation and admission control in the router environment instead of traffic characterization in the host environment as this paper mostly applies to.

Other related ones include the Renegotiated CBR service by M. Grossglauser, S. Keshav, and D. Tse [5] and similar work by S. Chong, S.Q. Li, and J. Ghosh [2] and H. Zhang and E.W. Knightly [17]. In their work, renegotiation is used to capture the slow time-scale variations with a stepwise-CBR rate allocation, while in this paper we further take advantage of the token bucket model and associated buffering to absorb the fast time-scale variations to allow for more statistical multiplexing gain in the network.

II. Traffic modeling using token bucket without queuing

In this section, we model a given traffic pattern by a token bucket so that there will be no packet loss or delay in the flow. This will be referred to the "no-delay requirement" in the rest of this paper. We identified properties of parameters b and r of such a token bucket, which include the bounds of valid values for them and relations between the optimal choices of them. These results not only applied to the no-queue case in this section, but also laid foundation and motivated the development of token bucket modeling in the queuing case presented in the next section.

A. Bounds for b and r

Intuitively, if r is too small, there will not be enough tokens generated in the bucket for incoming packets, no matter how large the capacity b is. Hence as a necessary condition to satisfy the no-delay requirement,

$$c_0 + r(t_k - t_0) \geq \sum_{i=1}^{k} p_i, \qquad \forall k \geq 1. \tag{1}$$

where c_0 is the initial token count in the token bucket at t_0. Hence

$$r \geq r_{\min} = \max_{1 \leq k \leq n} \frac{\sum_{i=1}^{k} p_i - c_0}{t_k - t_0} \tag{2}$$

On the other hand, there is no point making r infinitely large, either. The largest r we would ever need is when all tokens for incoming packets are freshly replenished. Generally,

$$c_0 + r(t_1 - t_0) \leq p_1 \tag{3}$$
$$r(t_k - t_{k-1}) \leq p_k, \quad k \geq 2. \tag{4}$$

Therefore, the maximal r is given by

$$r_{\max} = \max_{2 \leq k \leq n} \{\frac{p_1 - c_0}{t_1 - t_0}, \frac{p_k}{t_k - t_{k-1}}\}. \tag{5}$$

Lemma 1 *The values of $r_{\min}$ and $r_{\max}$ defined above satisfy $r_{\min} \leq r_{\max}$.*

Proof. If this is not true, let k be the index such that

$$r_{\min} = \frac{\sum_{i=1}^{k} p_i - c_0}{t_k - t_0} \tag{6}$$
$$> \max_{2 \leq j \leq n} \{\frac{p_1 - c_0}{t_1 - t_0}, \frac{p_j}{t_j - t_{j-1}}\}. \tag{7}$$

Then

$$(\sum_{i=1}^{k} p_i - c_0)(t_1 - t_0) > (p_1 - c_0)(t_k - t_0) \tag{8}$$

and

$$(\sum_{i=1}^{k} p_i - c_0)(t_j - t_{j-1}) > p_j(t_k - t_0), \quad \forall j \geq 2. \tag{9}$$

Summing up both sides of (8) and (9) for $2 \leq j \leq k$, we get

$$(\sum_{i=1}^{k} p_i - c_0)(t_k - t_0) > (\sum_{i=1}^{k} p_i - c_0)(t_k - t_0), \tag{10}$$

a contradiction. $\square$

Note that the peak rate used in TSpec is

$$p = \max_{1 \le k \le n} \frac{p_k}{t_k - t_{k-1}} \tag{11}$$

with $p \ge r_{\max}$ and the equality holds if $c_0 = 0$. When r reaches its maximum $r_{\max}$, b can take its minimum

$$b_{\min} = \max_{1 \le k \le n} (p_k, c_0). \tag{12}$$

For any $r \in [r_{\min}, r_{\max}]$, the maximal possible tokens accumulated before the arrival of packet p_k is

$$c_k(r) = c_0 + r(t_k - t_0) - \sum_{i=1}^{k-1} p_i. \tag{13}$$

The token count $c_k(r)$ can only be reached if the bucket size is big enough so that no token growth will be suspended by hitting the upper bound b before the arrival of p_k. Note $c_k(r) \ge p_k$ because $r \ge r_{\min}$. Hence the maximal b we ever need to consider for a given r is

$$b_{\max}(r) = \max_{1 \le i \le n} c_i(r). \tag{14}$$

Intuitively, $b_{\min} \le b_{\max}$. The optimal choice for b for a given r, i.e., the minimal b to satisfy the no-delay requirement, is denoted by $b_{\text{opt}}(r)$. We have $b_{\text{opt}}(r) \le b_{\max}(r)$, for all $r \in [r_{\min}, r_{\max}]$. We formulate the above results into a theorem.

Theorem 2 *For a given traffic pattern $\{(p_k, t_k) \mid 1 \le k \le n\}$ of a network flow on the time interval $[t_0, t_n]$, the valid range for the token bucket rate r is $[r_{\min}, r_{\max}]$ with the bounds given by (2) and (5). For any $r \in [r_{\min}, r_{\max}]$, the optimal token bucket size b satisfying the no-delay requirement must be between $b_{\min}$ and $b_{\max}(r)$ defined by (12) and (14) respectively. Moreover, we have the peak rate p given by (11) and the maximal packet size $M = \max_{1 \le k \le n} p_k$.*

B. Relationship between optimal b and r

For any given $r \in [r_{\min}, r_{\max}]$, if b is big enough (e.g., $b = b_{\max}$), the token bucket (r, b) should satisfy the no-delay requirement. In this section, we will find what the optimal, i.e., minimal such b is for a given r.

For $k = 1, 2, ..., n$, let's denote by $\Delta_k = t_k - t_{k-1}$ the inter-arrival time. The number of tokens in the token bucket at the time instant right before the arrival of the packet p_k is denoted by $c_k(r, b)$. The functions $c_k(r, b)$ are expressed recursively by

$$c_1(r, b) = \min(c_0 + r\Delta_1, b), \tag{15}$$
$$c_k(r, b) = \min(c_{k-1}(r, b) + r\Delta_k - p_{k-1}, b),\ k > 1. \tag{16}$$

Note here we use the same notation c_k to denote two different, but closely related functions $c_k(r)$ and $c_k(r, b)$ for simplicity. As a matter of fact, $c_k(r, b) \le c_k(r)$ and the equality holds for $b \ge b_{\max}$.

Our key concern here is the no-delay requirement $c_k(r, b) \ge p_k$, namely,

$$c_1 = \min(c_0 + r\Delta_1, b) \ge p_1, \tag{17}$$
$$c_k = \min(c_{k-1} + r\Delta_k - p_{k-1}, b) \ge p_k,\ k > 1. \tag{18}$$

To solve this system of n inequalities, we start with the last one which corresponds to $k = n$. This inequality says that

$$c_{n-1} + r\Delta_n - p_{n-1} \ge p_n, \tag{19}$$

and $b \ge p_n$. Combining (19) with (18) for $k = n - 1$, we get

$$c_{n-1} = \min(c_{n-2} + r\Delta_{n-1} - p_{n-2}, b) \tag{20}$$
$$\ge p_{n-1},\ p_n + p_{n-1} - r\Delta_n. \tag{21}$$

Hence, we obtain

$$b \ge p_n,\ p_{n-1},\ p_n + p_{n-1} - r(t_n - t_{n-1}), \tag{22}$$

and

$$c_{n-2} \ge p_{n-2} + p_{n-1} - r(t_{n-1} - t_{n-2}), \tag{23}$$
$$p_{n-2} + p_{n-1} + p_n - r(t_n - t_{n-2}). \tag{24}$$

Continuing this process inductively for $k = n-2, n-3, ..., 1$, we obtain the explicit solution to (17) and (18)

$$b \ge \sum_{i=u}^{v} p_i - r(t_v - t_u), \quad \forall\, 1 \le u \le v \le n. \tag{25}$$

Theorem 3 *For any $r \in [r_{\min}, r_{\max}]$, the optimal b, i.e., the minimal b to satisfy the no-delay requirement, is*

$$b_{opt}(r) = \max_{1 \le u \le v \le n} \Big(\sum_{i=u}^{v} p_i - r(t_v - t_u),\ c_0\Big), \tag{26}$$

if the initial token count c_0 is prescribed. If the token bucket is initially full, i.e., c_0 is in fact b, then for $r \in [r_{\min}, r_{\max}]$,

$$b_{opt}(r) = \max_{1 \le u \le v \le n} \Big(\sum_{i=u}^{v} p_i - r(t_v - t_u)\Big), \tag{27}$$

and in this case, we choose

$$r_{\min} = \frac{\sum_{k=1}^{n} p_k}{t_n - t_0}, \tag{28}$$

the average rate, and $r_{\max}$ to be the peak rate given by (11). From these expressions, we see that $b_{opt}(r)$ is a piecewise linear and decreasing function of r.

Remark. The solution (25) also follows from the following intuition. Immediately before the time t_u, the maximal possible token count is b. During the time interval $[t_u, t_v]$, at most $r(t_v - t_u)$ tokens are generated. Hence the token count is at most $b + r(t_v - t_u)$. But we need to consume $\sum_{k=u}^{v} p_k$ tokens to send all these packets out. Hence we must have

$$b + r(t_v - t_u) \geq \sum_{k=u}^{v} p_k. \quad \square \tag{29}$$

Now we have found all token buckets with which the given traffic pattern P is compliant. On the (r, b) plane, these token buckets are represented by the graph of the function $b = b_{\text{opt}}(r)$ on $[r_{\min}, r_{\max}]$. Later we will call this piecewise linear and decreasing curve the no-delay TB-curve for the pattern P, where TB stands for Token Bucket.

C. Algorithms and experimental results

According to Theorem 3, the value of b_{opt} is the maximal of at least $n(n-1)/2$ numbers. It would be too expensive to evaluate all these $O(n^2)$ numbers. Here we present two effective algorithms to compute $b_{\text{opt}}(r)$ for a given r. The first algorithm parses the input pattern P only once and is of complexity $O(n)$. The second one is a multi-pass algorithm and is of complexity $O(n \log n)$, which however can be generalized to the queuing case.

For the linear one-pass algorithm, we consider how to compute the sequence of

$$b_k = \max_{1 \leq u \leq v \leq k} \left(\sum_{i=u}^{v} p_i - r(t_v - t_u) \right), \; 1 \leq k \leq n, \tag{30}$$

recursively with respect to k. By Theorem 3, our $b_{\text{opt}}(r)$ is either $\max(b_n, c_0)$ or simply b_n itself, depending on if the bucket has a prescribed initial count c_0 or is initially full. For this purpose, we introduce an auxiliary notion

$$d_k = \max_{1 \leq u \leq k} \left(\sum_{i=u}^{k} p_i - r(t_k - t_u) \right), \; 1 \leq k \leq n. \tag{31}$$

Note that b_k is the maximum of $k(k-1)/2$ numbers, and each of these numbers corresponds to the flow on an interval $[t_u, t_v]$. The value b_k is the maximum of the subset of those numbers whose intervals end at t_k. Obviously, $d_1 = p_1$. If d_{k-1} is found, then d_k is either the previous maximum d_{k-1} plus the new input $p_k - r\Delta_k$ brought by the new (p_k, t_k) pair, or simply p_k, whichever is bigger.

After knowing how to compute d_k inductively, we are ready to solve b_k now. First $b_1 = p_1$. Assume b_{k-1} is found. The value of b_k as defined to be the maximum of $k(k-1)/2$ numbers is reached on an interval $[t_u, t_v]$. If $v < k$, $b_k = b_{k-1}$. Otherwise, $v = k$, then $b_k = d_k$. In other words, with the arrival of a new packet p_k at time t_k, the token bucket size either needs or needs not updating. The case that it needs to be updated is covered by the consideration of d_k. To summarize, we have the following inductive algorithm.

Theorem 4 (No-delay TB algorithm) *For a traffic pattern $P = \{(p_k, t_k) \mid 1 \leq k \leq n\}$ and a token bucket rate $r \in [r_{\min}, r_{\max}]$, the optimal size b of an initially full token bucket is the value of b_n given by the following recursive formulas.*

$$d_1 = b_1 = p_1, \tag{32}$$
$$d_k = \max(d_{k-1} + (p_k - r\Delta_k), p_k), \; k > 1, \tag{33}$$
$$b_k = \max(b_{k-1}, d_k), \; k > 1. \tag{34}$$

Remark. It is worth noting that the above algorithm is not only CPU efficient due to the $O(n)$ complexity, but also efficient in terms of memory. Indeed, one needs to allocate only two integers d and b as the placeholders for d_k and b_k sequences. There is no need to store the pattern data P since this is a one-pass algorithm, which can proceed in real-time along with the stream of arriving packets. $\square$

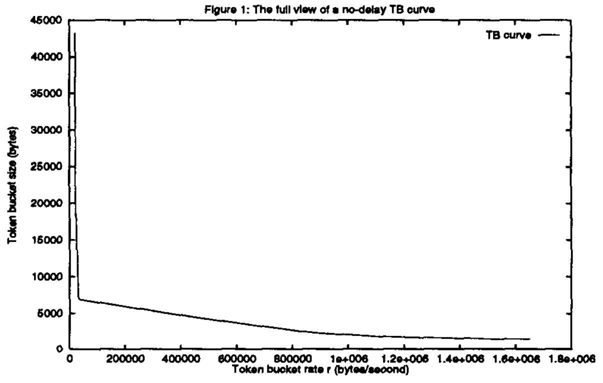


Figure 1: The full view of a no-delay TB curve

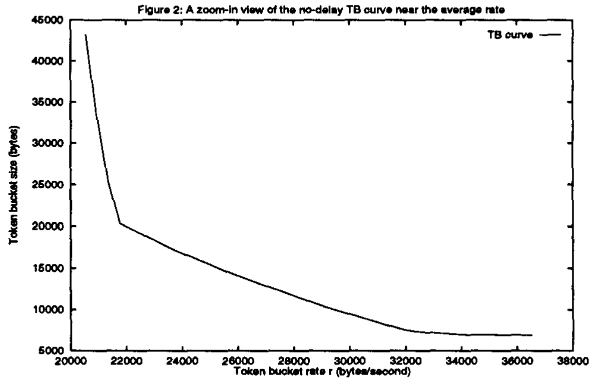


Figure 2: A zoom-in view of the no-delay TB curve near the average rate

We have implemented the no-delay TB algorithm in the MBTS software package. Here are the experimental results we have obtained with a 96,883.3 millisecond long H.263 video trace. This sample trace was captured from the movie "True Lies" using a kernel-mode packet scheduler in Microsoft Windows NT. It contains 1,664 packets and is 1,989,862 byte long. The token bucket is initially full. The average rate is $r_{\min} = 20,539$ bytes/seconds, and the peak rate is $r_{\max} = 1,652,500$ bytes/second. Figure 1 is the full view of the no-delay TB-curve of this trace. Figure 2 is a zoom-in view of the same curve around the average rate.

The second algorithm does not take advantage of the explicit solution for $b_{\text{opt}}(r)$. Instead, it does a binary search for the minimal b in its range such that $c_k(r,b) \geq p_k$ for all k. The merit of this algorithm is that it can be generalized and motivates the queuing case solution in section III. Also it serves as a reference to check the correctness and performance of the first algorithm in our experiments.

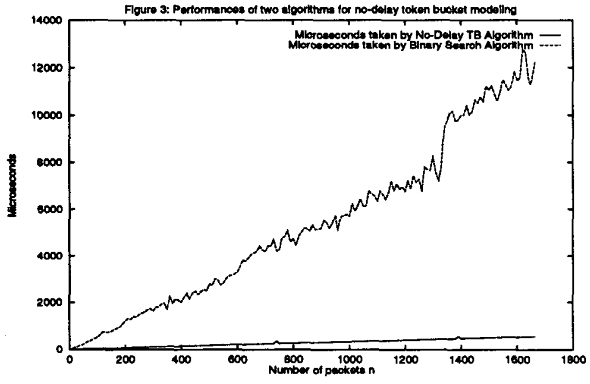
Let the range of b be denoted by $[b_l,\ b_u]$, where l stands for lower bound and u stands for upper bound. Set initially $b_l = b_{\min}$, $b_u = b_{\max}$, and $b = \frac{1}{2}(b_l + b_u)$. Here $b_{\min}$ and $b_{\max}$ are defined in (12) and (14) respectively, if c_0 is prescribed. In the case of initially full bucket, we choose $b_{\min} = M$ and $b_{\max} = \sum_{k=1}^{n} p_k \leq nM$.

We compute $c_k(r,b)$ inductively and compare its value with p_k. If $p_k \leq c_k(r,b)$ for all $k = 1, ..., n$, set $b_u = b$ and $b = \frac{1}{2}(b_l + b_u)$. Otherwise, when the first k with $p_k > c_k(r,b)$ is encountered, we set $b_l = b$ and $b = \frac{1}{2}(b_l + b_u)$. Repeat the steps above till $b_l = b_u$.

Since the initial range of b for the binary search is at most of size $O(n)$, there are exactly $O(\log n)$ updates on b. For each update of b, one needs to traverse the list of (p_k, t_k) at most once. In the worst case, this takes time $O(n)$. Hence the complexity of the binary search algorithm is $O(n \log n)$.

This algorithm has also been implemented in the MBTS software package. The experiments show that it produces exactly the same TB-curves as in Figure 1 and Figure 2, which confirms empirically the correctness of Theorem 3 and Theorem 4, in addition to the analytic proof given above. The first algorithm has much better performance than the second one. Figure 3 shows the time in microseconds taken to compute the optimal b value for a given r for n packets. These data were collected on a Pentium Pro 200 MHz PC with 64 MB of RAM. (There is a noticeable jump at $n = 1340$. This is due to locality change in memory. A linked list is used to store the traffic pattern. The link at 1340th packet is cross memory pages. This observation is confirmed by shifting location of the head of linked list. Similar jumps also exist on later figures about performance.)

Figure 3: Performances of two algorithms for no-delay token bucket modeling



Symmetrically, for any given $b \in [b_{\min},\ b_{\max}]$, we can use a similar procedure to find the minimal $r \in [r_{\min},\ r_{\max}]$ such that $c_k(r,b) \geq p_k$, for all k. This also defines a piecewise linear decreasing function $r = g(b)$. This function g is the inverse of $b_{\text{opt}}(r)$ on the set where $b_{\text{opt}}(r)$ is bijective.

All token bucket parameters, which can be used to accommodate a given traffic pattern, form a region in the (r, b) plain above the no-delay TB curve. Optimal choices reside on the lower boundary, i.e., the TB curve, of the region. With an extra cost function, for example, $r+b$, the standard optimization method can be used to determine an optimal token bucket model on the TB curve that minimizes the cost function. The tariff table used by a video server for accounting and billing purpose is another example of such cost functions.

D. Partridge and Garrett's Send-Now algorithm

As pointed out in section I.D Related Work, we will describe an undocumented and unpublished algorithm used by Craig Partridge and Mark Garrett in their "tb" program. This algorithm, called Send-Now, addresses the no-queue case in a way different from our No-Delay TB algorithm. It works only in the case of initially full token bucket.

Let c be the variable to denote the token count in the bucket immediately after sending p_k out. Then the Send-Now algorithm is as follows.

```
$c = b = 0$;
for ($k = 1$ to $n$)
        $c = \min(c + r\Delta t_k,\ b)$;
        if ($p_k \leq c$)
                $c = c - p_k$;
        else
                $b = b + (p_k - c)$;
                $c = 0$;
```

When a new packet p_k arrives, this algorithm first updates the token count c. If the updated count is big enough, it sends the packet out, and reduce c by p_k. Otherwise, it increases the initial token count, i.e., the token bucket size, by the credit shortage $p_k - c$. The subtle point here is that this increment in the initial token count is reserved all the way till p_k's arrival, i.e., will not be altered by credit growth cutoff in the middle. Since this increment in b is minimal to send p_k out, the resulting b is the minimal for the given r and traffic pattern.

We implemented this Send-Now algorithm, and found it produced exactly the same results as our No-Delay TB algorithm. The complexity of Send-Now and No-Delay TB algorithms are both linear, and both algorithms performs equally well for the traffic specification purpose. The No-Delay TB algorithm differs from Send-Now in the way that it is actually an inductive computation of the explicit solution (27). This explicit solution reveals generic properties of the TB curve, which are not obviously from the Send-Now algorithm. For example, if a cost function of $(r,\ b)$ is given, the explicit formula allows us to set up Euler-Langrange equation to solve the constrained optimization problem.

III. Traffic modeling using token bucket with a queue

The no-queue token bucket model is very sensitive to the changing behavior of a flow. A queue could be added to the token bucket to absorb short-term bursts of the flow, hence makes the token bucket modeling more stable. This section is devoted to the analysis of token bucket modeling in the queuing case.

A. Token bucket with a huge bucket size: a special case

We start with a special case where the token bucket has a big enough size b such that the token accumulation is never suspended. The situation is relatively simple due to the effect of b disappearing from the picture temporarily. Investigations under this assumption will not only give bounds for valid r and b, as in the no-queue case, but also provide insights to deal with more general situations.

Let's denote the queue size by q. If no packet loss is permitted at the token bucket, any valid values for r must satisfy

$$c_0 + r(t_k - t_0) + q \geq \sum_{t=1}^{k} p_i, \; \forall\, k \geq 1. \qquad (35)$$

More careful analysis given below shows that (35) is necessary for the no packet loss requirement, but not sufficient. A sufficient condition we would like to use is

$$c_0 + r(t_k - t_0) + q \geq \sum_{t=1}^{k} p_i + M, \; \forall\, k \geq 1, \qquad (36)$$

with $M = \max_{1 \leq i \leq n} p_i$. Therefore the minimal value $r'_{\min}$ considered for a token bucket rate in the queuing case is given by

$$r'_{\min} = \max_{1 \leq k \leq n} \frac{\sum_{i=1}^{k} p_i - c_0 - q + M}{t_k - t_0}. \qquad (37)$$

If we regard $r'_{\min}$ as a function $r'_{\min}(q)$ of $q \in [0, \infty)$, it is strictly decreasing and piecewise linear. Moreover, $r'_{\min}(M) = r_{\min}$.

The estimate (5) for the maximal value $r_{\max}$ needed for r still holds in this case. Same proof of Lemma 1 shows that $r'_{\min} \leq r_{\max}$ if $q \geq M$. This condition is natural because the queue size should be big enough to contain at least a single packet. Similarly, we still have $b_{\min} = \max_{1 \leq k \leq n} p_i$.

Denote by $b'_{\max}$ the value of the maximal number of tokens in a bucket when no token accumulation is cut off by reaching the upper bound b. This is the counterpart of $b_{\max}$ defined in (14) in the queuing case. Recall that we use $c_k(r)$ to denote the token count immediately before the *arrival* of p_k. We denote by $c'_k(r)$ the token count immediately before the *departure* of p_k. Then $b'_{\max}(r) = \max_{1 \leq k \leq n} c'_k(r)$. In the no-queue case, $c_k = c'_k$. This is no longer true in the queuing case. Hence we need to compute the departure time for each delayed packet before we can give $b'_{\max}(r)$. For any r, the token deficit immediately before the arrival of packet p_k, which is the number of tokens needed to send out all packets in queue, is expressed by

$$\sum_{i=1}^{k-1} p_i - c_0 - r(t_k - t_0). \qquad (38)$$

Let $I(k)$ denote the largest index $k' \leq k - 1$ such that

$$c_0 + r(t_k - t_0) - \sum_{i=1}^{k'} p_i \geq 0, \qquad (39)$$

i.e., $I(k)$ is the number of packets that have been sent out by the token bucket before the arrival of p_k. Then

$$c_k(r) = c_0 + r(t_k - t_0) - \sum_{k=1}^{I(k)} p_i. \qquad (40)$$

The number of bytes remaining in the queue immediately *after* the arrival of p_k is

$$q_k = \begin{cases} 0, \text{ if } c_0 + r(t_k - t_0) - \sum_{i=1}^{k} p_i \geq 0, \\ \sum_{i=I(k)+1}^{k} p_i, \text{ otherwise.} \end{cases} \qquad (41)$$

Proposition 5 *The condition $r \geq r'_{\min}$ guarantees $q_k \leq q$ for all k. Namely, the choice of the minimal token bucket rate will not cause any packet dropping.*

Proof. Assume $r \geq r'_{\min}$, then (36) is true. Rewrite it into

$$\sum_{i=I(k)+1}^{k} p_i \leq \left(c_0 + r(t_k - t_0) - \sum_{i=1}^{I(k)} p_i\right) + q - M \qquad (42)$$

$$= c_k(r) + q - M. \qquad (43)$$

The number of tokens $c_k(r)$ is not enough to send $p_{I(k)+1}$ out. Hence we get

$$\sum_{i=I(k)+1}^{k} p_i \leq p_{I(k)+1} + q - M \leq q. \qquad (44)$$

Therefore the number of bytes q_k need to be stored in queue does not exceed the queue size. □

Remark. Now we want to explain why (35) is not sufficient. Similar to (42), we rewrite (35) as

$$\sum_{i=I(k)+1}^{k} p_i \leq c_k(r) + q. \tag{45}$$

This only guarantees that $q_k \leq p_{I(k)+1} + q$, not $q_k \leq q$. Note here if we could discard $p_{I(k)+1}$ or break $p_{I(k)+1}$ into two parts, send out the part with $c_k(r)$ bytes first, and store the rest in the queue, then (35) would be sufficient. But neither fragmentation nor packet dropping is considered valid here, hence we have to raise the minimal token bucket rate to replenish tokens faster. We choose to add an extra term M as shown in (36), which basically adds more tokens to accommodate one more packet. □

So for a given queue size, we know how to find valid range for r now. On the other hand, if r, instead of q, is prescribed, the minimal queue size is

$$q = \max_{1 \leq k \leq n} q_k. \tag{46}$$

The delay for packet p_k is

$$d_k = \max\left(\frac{1}{r}\left(\sum_{i=1}^{k} p_i - c_0\right) - (t_k - t_0),\ 0\right). \tag{47}$$

The time to send the packet p_k out is delayed from t_k to t'_k, where

$$\begin{aligned} t'_k &= t_k + d_k \qquad (48) \\ &= \begin{cases} t_k, & \text{if } \dfrac{\sum_{i=1}^{k} p_i - c_0}{t_k - t_0} < r, \\ t_0 + \dfrac{\sum_{i=1}^{k} p_i - c_0}{r}, & \text{otherwise.} \end{cases} \qquad (49) \end{aligned}$$

Hence the pre-queuing traffic pattern $P = \{(p_i, t_k) \mid 1 \leq k \leq n\}$ is reshaped to the post-queuing traffic pattern $P' = \{(p_i, t'_k) \mid 1 \leq k \leq n\}$. An interesting observation is as follows.

Proposition 6 *The post-queuing traffic pattern P' is compliant with the token bucket $(b'_{\max}, r)$, and this r reaches the lower limit $r_{\min}$ defined by (2) with respect to P'.*

Proof. From (2) and (49), it is easy to see that

$$\frac{\sum_{i=1}^{k} p_i - c_0}{t'_k - c_0} = \begin{cases} \dfrac{\sum_{i=1}^{k} p_i - c_0}{t_k - c_0}, & \text{if it} < r, \\ r, & \text{otherwise.} \end{cases} \tag{50}$$

Therefore,

$$\max_{1 \leq k \leq n} \frac{\sum_{i=1}^{k} p_i - c_0}{t'_k - c_0} = r. \quad \square \tag{51}$$

Now with the presence of (49), we are ready to compute $b'_{\max}$, the upper bound for b in the queuing case. Note that

$$\begin{aligned} c'_k(r) &= c_0 + r(t'_k - t_0) - \sum_{i=1}^{k-1} p_i \qquad (52) \\ &= \begin{cases} c_0 + r(t_k - t_0) - \sum_{i=1}^{k-1} p_i, & \text{if it} \geq p_k, \\ p_k, & \text{otherwise.} \end{cases} \qquad (53) \\ &= \max\left(c_0 + r(t_k - t_0) - \sum_{i=1}^{k-1} p_i,\ p_k\right). \qquad (54) \end{aligned}$$

Hence

$$\begin{aligned} b'_{\max}(r) &= \max_{1 \leq k \leq n} c_k(r) \qquad (55) \\ &= \max_{1 \leq k \leq n} \left(c_0 + r(t_k - t_0) - \sum_{i=1}^{k-1} p_i,\ p_k\right). \qquad (56) \end{aligned}$$

Obviously, we have $b'_{\max}(r) \geq p_k$ for all k, hence $b'_{\max} \geq b_{\min}$. Now we have the queuing case counterpart of Theorem 2 with regard to the valid range of r and b.

Theorem 7 *For a given traffic pattern P and a queue size $q \geq M$, if the initial token count c_0 is prescribed, the valid range for the token bucket rate is $[r'_{\min}, r_{\max}]$. For any r in this range, we have the optimal, i.e., minimal token bucket size $b \in [b_{\min}, b'_{\max}(r)]$. Here M is the maximal packet size and the bounds are given by (37), (5), (12), and (56) respectively.*

Remark. In the case of initially full token bucket, i.e., $c_0 = b$, we choose $r_{\min}$ to be the mean rate in (28), $r_{\max}$ to be the peak rate in (11), $b_{\min}$ to be M, and $b_{\max} = \sum_{1 \leq k \leq n} p_k$. Note we have $b_{\max} \leq nM$. □

B. Token bucket parameters and queue size

In section II, we studied the no-queue case. In the previous subsection, we studied the queuing case without the restriction placed by b. Now we are ready to consider the general situation, namely, both the queue and token bucket size b come into play. Suppose we are given the values (r, b) residing below the no-delay TB curve, we want to know the minimal queue size for this token bucket and its corresponding post-queuing traffic pattern.

Again, we inductively define the function $c_k(r, b)$ which is the number of tokens existing in the bucket immediately before the arrival of p_k. Along with the definition of $c_k(r, b)$, the auxiliary index function $I(k)$ is reintroduced inductively for the general situation.

$$c_1(r, b) = \min(c_0 + r\Delta_1, b), \text{ and } I(1) = 0. \tag{57}$$

For $k \geq 2$, define $I(k)$ to be the largest index $k' \leq k - 1$ such that

$$c_{k-1}(r, b) + r\Delta_k - \sum_{i=I(k-1)+1}^{k'} p_i \geq 0, \tag{58}$$

namely, $I(k)$ is the number of packets that have been sent out *before* the arrival of p_k. Then define

$$c_k(r,b) = \min\left(c_{k-1}(r,b) + r\Delta_k - \sum_{i=I(k-1)+1}^{I(k)} p_i,\ b\right). \quad (59)$$

The number of bytes q_k need to be queued immediately *after* p_k's arrival is given by

$$q_k = \begin{cases} 0, \text{ if } I(k) = k-1 \text{ and } c_k(r,b) \geq p_k, \\ \sum_{i=I(k)+1}^{k} p_i, \text{ otherwise.} \end{cases} \quad (60)$$

Then the minimal queue size is

$$q(r,b) = \max_{1 \leq k \leq n} q_k. \quad (61)$$

This solves the first problem stated at the beginning of the section. Note also the function $I(k)$ is increasing in r and b, hence $q(r,b)$ is decreasing in r and b.

Now we consider two inverse problems. If the queue size is given, for any $r \in [r'_{\min}, r_{\max}]$, what is the optimal, i.e., minimal, choice for b? For any $b \in [b_{\min}, b_{\max}]$, what is the optimal choice for r? To answer this two questions, one only needs to solve the inequality

$$q(r,b) \leq q \quad (62)$$

for b or r with a minimal solution. In particular, we can express the optimal solution for b as a function of r, denoted by $b = b_q(r)$.

We expect to see that if $q = 0$, the optimal solution $b_0(r)$ to the inequality (62) is exactly the function $b_{\text{opt}}(r)$ we found in Theorem 3. Indeed, the definition (61) says that $q(r,b) = 0$ is equivalent to $c_k(r,b) \geq p_k$, for all k. Geometrically, on the (r, b)-plane, the graph of $b = b_q(r)$ is the level curve of the function $q = q(r,b)$ at the level q. We call this curve the queuing TB-curve with queue size q. The no-delay TB-curve is just the level curve of $q(r,b)$ at the level 0, i.e., the queuing TB-curve with queue size 0.

Since $q(r,b)$ is monotonously decreasing in both r and b, to compute the value of $b_q(r)$ for a given $r \in [r_{\min}, r_{\max}]$, we use binary search for the minimal $b \in [b_{\min}, b_{\max}]$ such that $q(r,b) \leq q$. In this procedure, the bounds for b or r identified in section III.A are used as initial values of the binary search. This algorithm is of complexity $O(n \log n)$.

This $O(n \log n)$ algorithm is a direct generalization of the second algorithm in section II.C. The only difference is that the inequalities $p_k \leq c_k(r,b)$ are replaced by $q_k \leq q$. How to find a linear, one-pass algorithm in the queuing case analogous to the no-delay TB algorithm is still an open problem.

The above binary search type algorithm has been implemented in the MBTS Software package. In the initially full bucket case, the initial range $[b_{\min}, b_{\max}]$ of b is usually very large, where $b_{\min} = M$, the maximal packet size, and $b_{\max} = \sum_{k=1}^{n} p_k \leq nM$. We added an optimization to the implementation by shrinking the initial range of the binary search. The optimization is to double the lower bound b_l of the range such that $q_k \geq q$ for all k. Then let the upper bound b_u be the value of this b_l and the lower bound be the half of this b_l. This optimization will be called "pre-bound optimization" since it's a precomputation about the upper and lower bounds of the binary search. We applied this algorithm to the aforementioned H.263 trace and obtained the following queuing TB-curves in Figure 4 with ten various queue sizes. Note that the TB-curve parameterized by $q = 0$ coincides with the no-delay TB-curve in Figure 1 and Figure 2. This confirms experimentally the consistency between the queuing TB curve with $q = 0$ and the no-delay TB-curve.

Figure 4: Queuing TB curves parametrized by the queue size

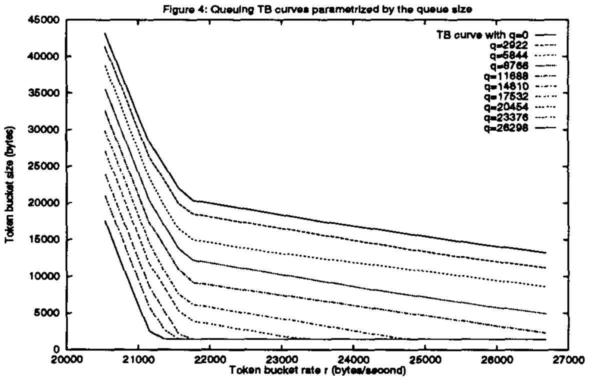
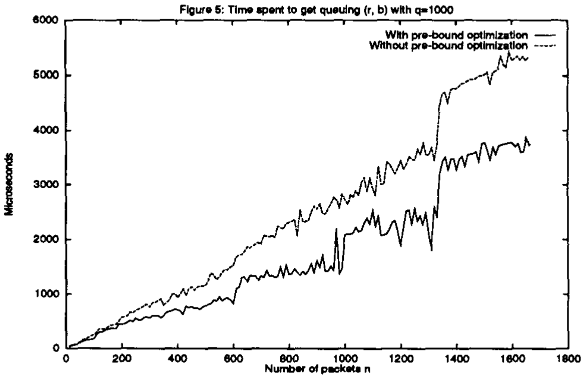


Figure 5 shows the computation time in microseconds taken by this algorithm as a function of number of packets n. The two curves represent the time taken by the algorithm with and without the optimization, respectively. The figure shows the optimization reduces the computation time by about 30%, which is due to the fact that the optimal b in the initial range $[b_{\min}, b_{\max}]$ is usually closer to $b_{\min}$.

Figure 5: Time spent to get queuing (r, b) with q=1000



C. Partridge and Garrett's Send-Smooth algorithm

In their "tb" program, Craig Partridge and Mark Garrett also considered the TB curve in the queuing case. But they restricted their consideration in very limited situation, i.e., they required a packet can only be delayed up to the point when the next packet arrives. Hence a queue with size of $M = \max p_k$ is enough to accommodate this situation.

Similar to section II.D, let c be the variable to denote the token count in the bucket immediately after sending p_k out. Let s be certain credit shortage when p_k needs be sent out. Then the Send-Smooth algorithm is as follows:

```
$c = b = 0$;
for ($k = 1$ to $n$)
        $c = c + \max(r\Delta t_{k+1} - p_k,\ 0)$;
        $c = \min(c,\ b)$;
        $s = \max(p_k - r\Delta t_{k+1},\ 0)$;
        if ($s \le c$)
                $c = c - s$;
        else
                $b = b + (s - c)$;
                $c = 0$;
```

The initial token count b is increased by the token deficit $s - c$, if any. They used the same trick as in the Send-Now algorithm that the increment in token count is passed all the way to the moment the deficit occurs. The advantage of this algorithm is that it is linear. But it only applies to a very special case.

D. Packet delays and post-queuing traffic pattern

Following section III.B, suppose we are given a queue with size q and a token bucket (r, b), we want to know the delay d_k, or, equivalently, the packet departure time $t'_k = t_k + d_k$, for each packet p_k as a result of queuing. We will derive a formula for t'_k and describe the post-queuing traffic pattern of the flow.

The time period when the token accumulation is suspended because the token count in the bucket reaches the bucket size is called the cut-off time. Let δ_k be the length of the cut-off time within $[t_{k-1}, t_k]$. Note $\delta_k \ge 0$ and the cut-off time is represented by a subinterval of $[t_{k-1}, t_k]$ ending at t_k. The subtle point here is that the cutting-off can start only after the packet $p_{I(k)}$ is sent out, because that $b \ge b_{\min}$. This point has actually been reflected in the expression (59). Recall $\Delta_k = t_k - t_{k-1}$. We have

$$c_0 + \sum_{i=1}^{k} r(\Delta_i - \delta_i) - \sum_{i=1}^{I(k)} p_i = c_k(r, b),\ k \ge 1. \qquad (63)$$

That implies the lengths δ_k of cut-off period, or equivalently, the lengths $\Delta_k - \delta_k$ of token growing period, are given by

$$\Delta_k - \delta_k = \frac{1}{r}\Big(\sum_{i=I(k-1)+1}^{I(k)} p_i + c_k - c_{k-1}\Big),\ k \ge 1. \qquad (64)$$

The expression (64) is intuitive in the following sense. During the period $[t_{k-1}, t_k)$, there are $c_{k-1}(r, b)$ tokens in the token bucket initially, $\sum_{i=I(k-1)+1}^{I(k)} p_i$ tokens are consumed by sending those $I(k) - I(k-1)$ packets out, and $c_k(r, b)$ tokens are leftover at the end. Hence the token count difference $\sum_{i=I(k-1)+1}^{I(k)} p_i + c_k(r, b) - c_{k-1}(r, b)$ must be provided by the token replenishing, that is, $r(\Delta_k - \delta_k)$.

To compute the delay of p_k, we introduce another auxiliary index function $J(k)$ which is either k if

$$c_0 + \sum_{i=1}^{k} r(\Delta_i - \delta_i) - \sum_{i=1}^{k} p_i \ge 0, \qquad (65)$$

or, otherwise, the largest index k' such that

$$c_0 + \sum_{i=1}^{k'} r(\Delta_i - \delta_i) - \sum_{i=1}^{k} p_i \le 0. \qquad (66)$$

Namely, p_k is sent out in the period $[t_{J(k)}, t_{J(k)+1})$. The condition (65) says that there is no delay for p_k. Otherwise, p_k is sent out at exactly the moment $t'_k = t_{J(k)} + \tau \in [t_{J(k)}, t_{J(k)+1})$, for some $0 \le \tau < \Delta_{J(k)+1}$. Here τ can be obtained from

$$c_0 + \sum_{i=1}^{J(k)} r(\Delta_i - \delta_i) + r\tau - \sum_{i=1}^{k} p_i = 0. \qquad (67)$$

Therefore,

$$\tau = \frac{1}{r}\big(\sum_{i=1}^{k} p_i - c_0\big) - \sum_{i=1}^{J(k)} (\Delta_i - \delta_i). \qquad (68)$$

Before continuing to compute t'_k, we present the following interesting relations between the two auxiliary index functions.

Proposition 8 *The two monotonously increasing functions I and J satisfy*

$$I(k) < k \le J(k). \qquad (69)$$

if there is no delay for p_k, $I(k) = k - 1$ and $J(k) = k$. Moreover, they are approximate inverses of each other in the sense that

$$I(J(k)) < k, \quad I(J(k) + 1) \ge k, \qquad (70)$$

and

$$J(I(k)) < k, \quad J(I(k) + 1) \ge k. \qquad (71)$$

Proof. The inequality (69) and the statement about no-delay case follow from definitions of I and J immediately.

By definition, the k-th packet is sent out no earlier than time instant $t_{J(k)}$, hence less than k packets have been sent out before $t_{J(k)}$, or, $I(J(k)) < k$. Note the k-th packet is sent out before $t_{J(k)+1}$, so $I(J(k) + 1) \ge k$.

Note that $I(k)$-th packet is sent out before t_k, be definition of J, we have $J(I(k)) < k$. But $p_{I(k)}$ is the last packet sent out before t_k. So the next packet $p_{I(k)+1}$ is sent out no earlier than t_k. That implies $J(I(k)+1) \geq k$. $\square$

Remark. From the definition of J, it seems we need to use c_0, p_i, t_i, and δ_i to compute values of J. That would be messy from implementation point of view. But the above proposition tells us that

$$J(k) = \max\{k' \mid I(k') < k\}. \tag{72}$$

Hence we can easily derive J from I. The symmetric statement, though may not be as useful for implementation as the previous one, is also true, i.e.,

$$I(k) = \max\{k' \mid J(k') < k\}. \quad \square \tag{73}$$

According to (63), we have

$$c_0 + \sum_{i=0}^{J(k)} r(\Delta_i - \delta_i) - \sum_{i=1}^{I(J(k))} p_i = c_{J(k)}(r, b). \tag{74}$$

Continuing from (68),

$$\tau = \frac{1}{r}\Big(\sum_{i=1}^{k} p_i - \sum_{i=1}^{I(J(k))} p_i - c_{J(k)}(r, b)\Big) \tag{75}$$

$$= \frac{1}{r}\Big(\sum_{i=I(J(k))+1}^{k} p_i - c_{J(k)}(r, b)\Big). \tag{76}$$

The intuition shown in (76) is described as follows. The packet $p_{I(J(k))+1}$ is the first one sent out after (more precisely, no earlier than) $t_{J(k)}$. To send out all packets p_i, where $I(J(k)) + 1 \leq i \leq k$, the token bucket needs $\sum_{i=I(J(k))+1}^{k} p_i$ tokens, while $c_{J(k)}(r, b)$ tokens exist immediately before $t_{J(k)}$. Hence $\tau = \big(\sum_{i=I(J(k))+1}^{k} p_i - c_{J(k)}(r, b)\big)/r$ units of time are needed to grow enough tokens. The way $J(k)$ is chosen guarantees that so computed $\tau \in [0, \Delta_{J(k)+1})$.

Recall (76) is obtained in the case that p_k is delayed. Combining the no-delay case, we get the general formula

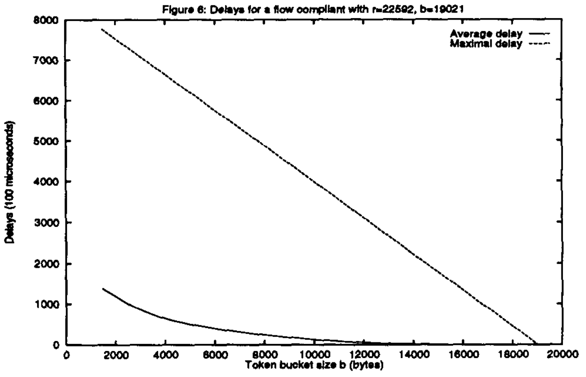
$$\tau = \max\Big(\frac{1}{r}\big(\sum_{i=I(J(k))+1}^{k} p_i - c_{J(k)}(r, b)\big),\, 0\Big). \tag{77}$$

Theorem 9 *If a queue with size q is used to store the flow with the traffic pattern $P = \{(p_k, t_k) \mid 1 \leq k \leq n\}$, and r and b are token bucket parameters satisfying $q(r, b) \leq q$, the post-queuing traffic pattern will be $P' = \{(p_k, t'_k) \mid 1 \leq k \leq n\}$ with t'_k given by*

$$t'_k = t_{J(k)} + \max\Big(\frac{1}{r}\big(\sum_{i=I(J(k))+1}^{k} p_i - c_{J(k)}(r, b)\big),\, 0\Big). \tag{78}$$

A direct application of the theorem is to compute delay of each packet caused by queuing, hence the average delay and maximal delay of the flow. Figure 6 shows the average delay and maximal delay as a function of b as r is fixed. The sample flow is still the same H.263 trace from the movie "True Lies".

Figure 6: Delays for a flow compliant with r=22592, b=19021

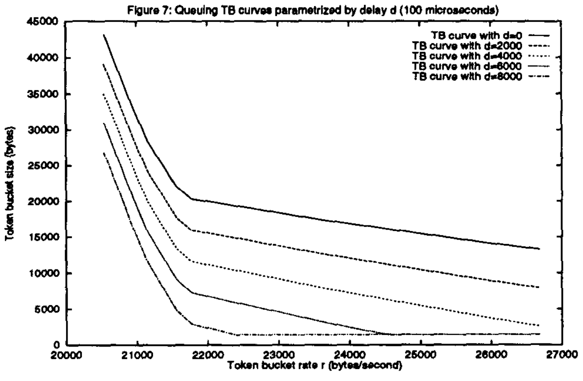


Note that for a fixed r, both maximal delay and average delay are decreasing functions of b. This property enables us to answer the inverse problem of the above theorem. The inverse problem is that if an upper bound d of the maximal delay (or average delay) is prescribed for a pre-queuing pattern P, what the token bucket parameters and minimal queue size one should use.

To solve this problem for the maximal delay, we do binary search again for the minimal b in its range for a fixed r such that all delays $d_k \leq d$, similar to what we did in section III.B. The only difference is that the inequalities $q_k \leq q$ over there is replaced by $d_k \leq d$ here. Again, this algorithm is of the complexity $O(n \log n)$. To find a linear algorithm for this solution is also open. The problem for the average delay could be solved similarly.

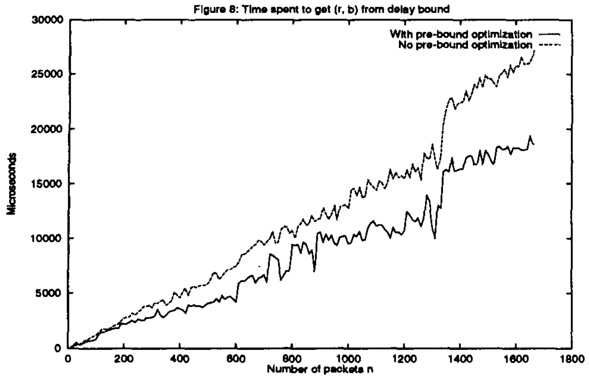
We applied the above algorithm to the sample H.263 trace. Figure 7 shows five TB curves on the (r, b)-plane corresponding to five different d values. Note the curve parameterized by $d = 0$ is consistent with the curve in Figure 4 parameterized by $q = 0$.

Figure 7: Queuing TB curves parametrized by delay d (100 microseconds)



In the implementation of this computation in the MBTS software package, we also applied the pre-bound optimization introduced in section III.B. Figure 8 presents the performances of this computation with and without the optimization, respectively. The empirical data shows the optimization reduces the computation time by about 27%.

Figure 8: Time spent to get (r, b) from delay bound



IV. Conclusion

The major contribution of this paper is the derivation of optimal token bucket parameters on the fly while observing the network flow. Relationships among observed traffic patterns, token bucket parameters, queue sizes, and queuing delays are analyzed quantitatively and presented with explicit formulas and/or efficient algorithms. These explicit formulas could be combined with other cost functions to derive the optimal operating point in the TB curve.

These theoretical results enable the measurement-based traffic specification (MBTS) technology, which characterizes network traffic adaptively in real-time and allocates network resources transparently on behalf of the application. Experiments using our implementation of this technology confirmed the analytical results and showed that the amount of time spent on the computation of the token bucket parameters is negligible.

We expect MBTS technology would ease the use of RSVP and the like, hence speed up the deployment of Internet QoS services. Furthermore, the precision and adaptability inherent in MBTS would help network applications subscribe the appropriate amounts of network resources, hence improve the network efficiency.

Acknowledgement

We appreciated the valuable discussions with Craig Partridge regarding the algorithms used in their "tb" program.

References

[1] B. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin *Resource Reservation Protocol (RSVP) - Version 1 Functional Specification*, RFC 2205, September 1997

[2] S. Chong, S.Q. Li, and J. Ghosh *Predictive Dynamic Bandwidth Allocation for Efficient Transport of Real-Time VBR Video over ATM*, IEEE Journal on Selected Areas in Communications, Vol. 13, No. 1, Jan. 1995

[3] D. Clark, S. Shenker, and L. Zhang *Supporting real-time applications in an integrated services packet network: Architecture and mechanism*, Proc. ACM SIGCOMM '92, 1992

[4] M. W. Garrett and W. Willinger *A Framework for Robust Measurement-Based Admission Control*, Proc. ACM SIGCOMM '94, pages 269-280, 1994

[5] M. Grossglauser, S. Keshav, and D. Tse *RSBR: A Simple and Efficient Service for Multiple Time-Scale Traffic*, Proc. ACM SIGCOMM '95, page 219-230, 1995

[6] M. Grossglauser and D. Tse *Analysis, Modeling and Generation of Self-Similar VBR Video Traffic.*, Proc. ACM SIGCOMM '97, pages 237-248, 1997

[7] S. Jamin, P.b. Danzig, S. Shenker, and L. Zhang *A Measurement-Based Admission Control algorithm for Integrated Services Packet Networks*, Proc. ACM SIGCOMM '95, 1995

[8] E. Knightly and H. Zhang *Traffic Characterization and Switch Utilization using a Deterministic Bounding Interval Dependent Traffic Model*, Proc. IEEE INFOCOM '95, Boston, Mass., April 1995

[9] O. Ohnishi, T. Okada, and K. Noguchi *Flow Control Schemes and Delay/Loss Tradeoff in ATM Networks*, IEEE Journal of Selected Areas in communications, 6(9):1609-1616, Dec. 1988

[10] C. Partridge *Gigabit Networking*, Addison Wesley Publishers, 1994

[11] C. Partridge *Manual page of TB program*, 1994

[12] S. Shenker, C. Partridge, and R. Guerin *Specification of Guaranteed Quality of Service*, RFC 2212, September 1997

[13] S. Shenker and J. Wroclawski *General Characterization Parameters for Integrated Services Network Elements*, RFC 2215 September 1997

[14] P. Tang *Interface Specification of Measurement-based Traffic Specifier*, available from Perry.Tang@intel.com, 1997

[15] *Windows Socket 2 Application Programming Interface, An interface for transparent network programming under Microsoft Windows*, Revision 2.2.1, May 2, 1997

[16] J. Wroclawski *Specification of the Controlled-Load Network Element Service*, RFC 2211, September 1997

[17] H. Zhang and E.W. Knightly *A New Approach to Support Delay-Sensitive VBR Video in Packet-Switched Networks*, Proc. 5th Workshop on Networking and Operating Systems Support for Digital Audio and Video, pages 275-286, April 1995