-- --

DYNAMIC TIME WINDOWS: PACKET ADMISSION CONTROL WITH FEEDBACK

*Theodore Faber * and Lawrence H. Landweber ***
Department of Computer Sciences University of Wisconsin - Madison
faber@cs.wisc.edu and lhl@cs.wisc.edu

Amarnath Mukherjee
Computer and Information Science Department University of Pennsylvania
amarnath@central.cis.upenn.edu

Abstract

We present a feedback congestion control method, Dynamic Time Windows, for use in high speed wide area networks based on controlling source variance. It is part of the two-level integrated congestion control system introduced in our earlier work[1].

The method consists of a packet admission control system and a feedback system to dynamically control source burstiness. Source throughput is not modulated as with traditional packet windows, allowing system throughput to remain high while avoiding congestion. Furthermore, the admission control bounds congestion times in the network, allowing feedback to be effective in the face of large bandwidth delay products.

The basic control mechanisms are analogs to traditional packet windows applied to controlling time windows - a new mechanism which allows switches to modulate source variances. The proposed system is simulated, and the results reported and analyzed. Enhancements to the basic system are also proposed and analyzed.

We wish to stress that the system described here is the second level of a two-level congestion control. Previous work[1] concentrated on the switch queueing mechanism, Pulse, while this work is a detailed examination of the feedback system used to adjust time windows to changing network load.

1. Introduction

This paper is a description of a new congestion control method called Dynamic Time Windows (DTW). DTW is the admission control portion of a two level congestion control system, the other portion of which is the Pulse queueing discipline, a generalization of Virtual Clock[2]. Our previous work[1] develops the Pulse queueing discipline and Generalized Virtual Clocks in detail. This paper is a detailed investigation of the feedback system used to manipulate the time windows.

DTW is intended to operate in a high speed wide area network (HSWAN). This is a network with line speeds on the order of one Gigabit/second and correspondingly high bandwidth-delay product. In this environment we are concerned with classical data source that is sensitive to loss, but less sensitive to delay. Examples of this sort of traffic include file servers, and servers in client-server models of distributed computation, like RPC servers. Other examples are processes monitoring experiments and sending data to other sites for analysis, seismic monitoring, or collection of high speed computer performance traces.

The network environment that DTW is intended for presents some unique challenges. Most important is the high bandwidth delay product which hampers the effectiveness of traditional feedback-based congestion control systems. Also, even though we concentrate on data traffic, a network the size and speed of the one envisioned will have sources with different burstiness characteristics. This range of traffic behavior implies that the control mechanisms must be simple and robust.

The network is further assumed to be reservation-based and virtual circuit-oriented, allowing sources to specify their traffic patterns when they establish their connections. DTW takes the traffic specifications as a contract between the source and the network, such that as long as the source remains within the negotiated bounds, the network should provide it with good service. Our goal is to provide a service that maximizes useful throughput for data traffic that obeys the traffic description that it negotiates with the network.

Dynamic Time Windows meets the challenges of providing congestion control based on controlling the burstiness of sources. The *time window* can be considered as the interval over which a source's average rate is computed. For example a source averaging 100 Mb/sec can be burstier if that average is computed over 1 second, than if the average is computed over 10 msec. By constraining sources to behave as if the average rate they negotiate with the network is computed over the current time window, we can control the variance of source sending rates. Our earlier work has shown that manipulation of the variance is effective in controlling queue length [1], and this is the knob that DTW uses to control congestion.

DTW provides a feedback mechanism whereby the network dynamically determines the time window size for the current network state, and adapts to changes in the state. Since the network parameter manipulated by DTW is burstiness and not outstanding packets, source throughputs are virtually unaffected by DTW. In this paper we explore algorithms to adjust the time window to adapt to a changing network state. We provide both analytic results and simulation studies.

The packet admission control used by DTW to constrain source behavior also has effects which mitigate the effects of the high bandwidth-delay product on feedback. Forcing sources to follow DTW rules causes congestion times to be bounded. This allows congestion to clear more easily and regularly enabling the feedback system to concentrate more on preventing new congestion than on clearing old congestion.

The basic DTW feedback algorithms have been designed to be integrated with the Generalized Virtual Clock algorithms presented in an earlier paper[1]. We believe congestion that remains for less than one round trip time, called short-term congestion, to be an issue best solved by queueing disciplines.

*This work was supported by an IBM Graduate Fellowship.
**This work was supported by the DARPA/NSF Gigabit Testbed Program via CNRI, and the Bell Laboratories XUNET program.

We assume the use of some enlightened switch queuing discipline, for example Virtual Clock[2], or Pulse[1]. This paper will concentrate on the packet admission control used to control source variance, and the feedback control used to modulate the time window to adapt to medium-term congestion.

The paper is organized as follows: Section 2 describes the network model in detail and gives definitions, Section 3 describes DTW admission control, Section 4 describes the feedback mechanisms, Section 5 describes simulation results. Section 6 discusses future directions of the research, and Section 7 draws conclusions.

2. Network Model and Definitions of Terms

2.1. Network Model

The network assumed by DTW is a network of switches interconnected by lines with a bandwidth on the order of one Gigabit/sec. All lines are point-to-point full duplex.

Switches are ''smart'' packet switches, meaning that they are assumed to be able to monitor their internal state, implement non-trivial queueing algorithms, and do simple operations related to DTW. Each switch is assumed to be capable of injecting packets into the network. Such packets are used to control the time windows of various sources. An example of such a switch is the XUNET switch[3].

The network contains data sources and sinks, which are senders and receivers of packets, and are called by the generic term *nodes*. A node can be both a source and a sink, but cannot be a switch. A node may source or sink as much data as the lines connected to it can carry. Nodes are responsible for assuring that data that flows from them obeys the DTW rules.

The network uses virtual circuits (VCs). A VC can exist between two nodes, one source and one sink, and must pass through at least one switch to be of interest. Each VC has associated service parameters, which are negotiated when it is set up. All packets in the network belong to some virtual circuit, and VCs are the basis for assigning resources, for example, switch buffers, to conversations in the network. All packets in the same VC follow the same path through the network. VCs are simplex, so a conversation actually consists of two one-way VCs connecting the same two nodes in opposite directions.

In this paper we only consider VCs passing through one switch so that we can more simply show the dynamics of the method. Analysis of multi-hop paths is underway.

2.2. Parameters of Dynamic Time Windows

As mentioned in Section 1, *time window* is the name given to the construct used to control the variance of a bursty source. Each VC has a time window associated with it, specified at set up time and varied dynamically for the life of that VC. The time window, combined with other parameters of the source forms our burstiness control.

When a source sets up a VC under DTW, it supplies three pieces of information in addition to the obvious routing and addressing information. The three pieces of information are the average rate at which it will send ($\bar{\lambda}$), the peak rate it will send at ($\hat{\lambda}$) and the time window ratio (*r*). The first two parameters are straightforward, but *r* requires some explanation. A switch keeps track of the maximum time window (MTW) it will allow a source with traffic passing through it to allocate; the size of the MTW is called $I_{\max}$. Each source multiplies $I_{\max}$ by *r* to arrive at the time window for that VC. The size of a given time window is denoted by *I*. All ratios are between zero and one, including one but excluding zero. The ratio is a relative burstiness factor where the most bursty source has a ratio of 1.0 and as the ratio gets smaller, the sources are less bursty. Another way to view *r* is that each source is getting some share of all the burstiness that the switch can accommodate. In either case, it is hoped that expressing burstiness as a pure number will help users more easily and accurately express their application's burstiness. $\bar{\lambda}$ and $\hat{\lambda}$ are given in bits/second or Mbits/sec. *I* is in seconds or msec.

The time window can be thought of as the interval over which a source's sending rate is averaged when computing the average rate. The basic rule for time window flow control is that over *I* time units only $\bar{\lambda}I$ bits of data may be sent, *i.e.*, over any *I* time units the average sending rate of a source is $\bar{\lambda}$. This leads to the derived quantity $\hat{t} = \bar{\lambda}I/\hat{\lambda}$, the amount of time a source can send at its maximum rate in any time window.

We constrain $I_{\max}$ to be greater than the largest source to switch delay of any source passing through the switch. The reason for this is that since time windows deal with medium-term congestion, making them overly fine controls would be infringing on the controls offered by the queuing discipline, and could lead to destructive interference.

3. Time Windows Admission Control

3.1. Previous Work

This work owes much to the insights displayed in other admission methods for congestion control in high speed networks. However, DTW is unique in that it is the only such method that incorporates feedback to adapt to a changing network. With the Leaky Bucket[4], and its variants[5],[6] a pool of tokens is maintained, where each token allows a packet transmission, but is consumed in the process. The tokens can be further characterized to indicate different loss priorities of traffic[5]. The name comes from the fact that the pool is being filled at a constant rate by tokens up to a fixed maximum, at which point they leak out. The rate of token input is based on the average sending rate and packet size. The accumulation of tokens during idle periods allows freedom to bursty sources. DTW can be thought of as a leaky bucket where the rate of token restoration is based on the sending pattern of the source over the previous time window.

Lixia Zhang presents another admission control method called a User Behavior Envelope[2]. Her goal is to keep a running total of source packets sent over each continuous averaging interval to insure that source variance remains finite. We use a similar method to assure that sources obey their negotiated DTW parameters.

Other admission controls are similar in spirit to DTW. Distributed Source Control[11] is an example; however, DTW's time windows are both overlapping, and dynamically adjusted to adapt to changing network load, unlike Distributed Source Control's smoothing intervals. Also we do not adjust individual source windows on a burst level. Stop-and-go queuing [12] and its attendant framing strategies provide bounded delays and jitter, but use a non-work conserving queueing discipline. These bounds are based upon a framing strategy that bears some resemblance to time windows, but are also non-overlapping. DTW does not provide such bounds, but uses a work conserving switch queueing discipline.

3.2. Basic Concepts

The primary rule of time windows is that over any time interval of length *I*, only $\bar{\lambda}I$ bits can be sent. This is a rate-based rule in the sense that a packet does not wait for an acknowledgement of a previously sent packet, but rather is released for sending based on the passage of time and the DTW parameters of the VC. It does not constrain the source's throughput in the way that conventional windows do. The source is always entitled to send enough bits in a time window to maintain its average sending rate. However, if the source needs to send a burst at its maximum rate, it can send for at most $\hat{t}$ time units.

The term *time window* is used because time windows are analogs of packet windows. Both types of window control a source's sending behavior by limiting when that source can send packets. A packet window constrains the number of unacknowledged packets that may be outstanding; a time window constrains the times when a packet may be sent. Packet windows are opened or advanced by the receipt of acknowledgements; time windows are opened by the passage of time. One can view a packet window as a window in sequence number space which determines how many packets can be outstanding, and a time window as a window in potential sending times which determines when a given packet can be sent.

The constraint above is the only one placed on sources, so they can have very general sending patterns. If we consider a source such that its sending rate at time t is given by $\lambda(t)$, we can express the time window requirement as:

$$\forall t, \quad \int_{t}^{t+I} \lambda(\tau)d\tau \le \bar{\lambda}I \qquad (3.1)$$

3.3. Time Independence

When using time windows for congestion control, it is crucial that the time windows requirement be met over *any* interval of length I, not just non-overlapping intervals. To insure this we use a credit method, similar to a Leaky Bucket[4]. Credits are in terms of bits. If a source has as many credits as bits in the packet it is ready to send it can send that packet, otherwise it must wait until it does, or send only as many bits of the packet as it has credits. Initially there are $\bar{\lambda}I$ credits in the credit pool, but when a credit is used, it is restored I time units later, unlike a leaky bucket where the credits are restored at the average sending rate.

This method can be easily implemented with one counter to hold the number of bits that have been sent, and a credit priority queue where each entry is time stamped, and maintained so that the most recent time stamp is at the head of the queue. Whenever a packet is to be sent, its length is added to the counter, and if that is less than $\bar{\lambda}I$, then that many credits are put on the credit queue with a timestamp I time units from now and the packet is sent, otherwise the packet is queued. Every time a packet is to be sent, the credit queue is first checked and any entries with timestamps earlier than the current time are subtracted from the counter, and then the packet is sent as above. If there are not enough credits, the credit queue can be consulted to determine when there will be enough, and the sending process put to sleep until then. Using a record of the bits sent in the last I time units allows a more natural implementation of opening and closing the time window, and changes in I are why the queue must be a priority queue. The value of $\bar{\lambda}I$ changes when the time window opens or closes, but this method insures correct behavior as long as the sending counter is forced to remain positive.

This algorithm performs the same function as Zhang's User Behavior Envelope[2], but we have modified the terminology and implementation to allow the source's time window to vary, thereby making DTW admission control part of a dynamic control system.

3.4. Time Window Admissions and Control Delay

One of the most interesting and useful attributes of time window admissions is that it guarantees that switch queues will empty regularly under some very loose requirements. We state the property as a theorem.

Theorem 1

Given that:

1. The switch will be serving packets at a rate μ whenever the queue is non-empty.
2. All time window ratios (r) are of the form $1/n$ for some integer n
3. The switch service rate is greater than or equal to the sum of the average rates of the sources sending packets through it.
4. k sources feed into a switch, with each having and obeying time windows parameters $\hat{\lambda}_k$, $\bar{\lambda}_k$, and r_k. A source's instantaneous sending rate is $\lambda_k(t)$. $I_k = r_k I_{\max}$ where $I_{\max}$ is maintained by the switch.
5. Due to the time windows criterion (equation (3.1))

$$\forall t,k, \quad \int_{t}^{t+I_k} \lambda_k(\tau)d\tau \le \bar{\lambda}_k I_k \qquad (3.1a)$$

6. We assume that $I_{\max}$ remains constant over the period in question.

Then: If a switch queue transitions from an empty state to having a non-zero queue length at time t, it will again be empty at or before time $t+I_{\max}$.

Proof

We proceed by contradiction, assuming that the queue will not have a zero queue length in the time period $[t, t+I_{\max}]$. Since the queue length was zero at t and by assumption the queue length is non-zero until time $t+I_{\max}$, the queue length at time $t+I_{\max}$ will be:

$$\left(\sum_{i=1}^{k} \int_{t}^{t+I_{\max}} \lambda_i(\tau)d\tau \right) - I_{\max}\mu \qquad (3.2)$$

$$= \left(\sum_{i=1}^{k} \left(\sum_{j=1}^{1/r_i} \int_{t+(j-1)r_i I_{\max}}^{t+jr_i I_{\max}} \lambda_i(\tau)d\tau \right) \right) - I_{\max}\mu \qquad (3.3)$$

Since each of the integrals in the inner sum of (3.3) is over a period of length I_k, (3.1a) yields the following:

$$\le \left(\sum_{i=1}^{k} \left(\sum_{j=1}^{1/r_i} \bar{\lambda}_i I_i \right) \right) - I_{\max}\mu \qquad (3.4)$$

$$= \left(\sum_{i=1}^{k} \frac{1}{r_i} \bar{\lambda}_i I_i \right) - I_{\max}\mu \qquad (3.5)$$

and since $I_i = I_{\max} r_i$

$$= I_{\max}\left(\sum_{i=0}^{k}\bar{\lambda}_i - \mu\right) \qquad (3.6)$$

Since $\mu \geq \sum_{i=1}^{k}\bar{\lambda}_i$ by Assumption 3, the queue length at $t + I_{\max}$ must be zero or less, which contradicts the assumption that it was non-zero at that time.

□

If we relax assumption 2, the theorem remains true, except that the bounding time becomes the least common multiple of the individual sources' time windows. Since arbitrary values of r can imply an arbitrarily long time bound, we usually constrain r to be an inverse integer so that the upper bound on congestion times is $I_{\max}$. Any constraint that bounds the least common multiple of the time windows is sufficient to bound the *renewal interval*.

This property of time window admissions reduces the effects of the high latency of control messages on congestion in the network. Consider a switch in a normal packet window-based congestion control method, like the DECbit algorithm [7], as it becomes congested. In the worst case, its queue length is steadily growing and will continue to grow until the control messages it is sending in acknowledgements reach the sources that are flooding it. The severity of the congestion, specifically the queue length at the bottleneck switch, is directly related to the time it takes for the control messages to be received[13]. Then the network must still remove the excess buildup to a safe level in the face of repeated retransmissions, which may cause cyclic congestion themselves. This reduces the effectiveness of the congestion messages once they arrive.

A switch congested in a DTW-based network will always have a zero queue length one MTW duration ($I_{\max}$) from when the congestion started. MTWs are required to be longer than the switch to source time, so this means that there is a high likelihood that the corrective action taken by DTW will have taken effect before the next congestion peak. Even if the control has not taken effect, there is no unbounded buildup of the queue length to make the congestion worse; the time that the switch is congested is always bounded. So when a new value of $I_{\max}$ takes effect, the switch queue length will become zero before the next $I_{\max}$ time units have expired. In order to ensure congestion relief, all that must be certain is that the new value of $I_{\max}$ is sufficient to prevent new congestion from occurring. Simulations bear this out. Notice that the congestion will temporarily clear itself even if no control message appears. Control messages now only play the role of preventing congestion and packet loss from recurring.

Another issue raised by the bounded congestion times is that the amount of buffering needed in the network to avoid packet loss altogether can be derived, based on the worst case sending pattern of sources. The worst case sending behavior of a source occurs when it sends all its packets immediately for $\hat{t}$, and the worst case for the switch occurs when all sources do this simultaneously. For example, the amount of buffering needed for 20 sources having identical parameters of $\bar{\lambda}$ = 50 Mb/sec, $\hat{\lambda}$ = 100 Mb/sec and r, and limiting the maximum of the source time windows to be 1.0 sec requires 500 Megabits of buffering for a 1 Gigabit/sec switch, and this buffering grows linearly as the MTW increases. Although calculating the amount of buffering needed to avoid packet loss is always possible, it is not always practical to provide sufficient buffering.

The other option is to reverse the equation and limit the MTW based on available buffering. However, in a case where sources are not behaving in the worst possible manner, which is the common case, this may place undue constraints on their operation and dramatically underuse switch resources. For this reason, optimistic booking of the switch resources and feedback are more appropriate control mechanisms.

4. Dynamic Time Windows and Feedback

This section begins by describing the basic feedback model used for DTW control, as well as performance enhancements. Throughout this section and Section 5 the term *switch queue length* will refer to the sum of all packets in service or awaiting service at the queue. If the switch is using a per-VC queuing scheme, like Pulse or another Generalized Virtual Clock [1], the switch queue length is the sum of all the per-VC queues.

In keeping with our two-level view of congestion control, we believe that the policing of flows is a function of the switch queuing discipline. Switch disciplines like Virtual Clock have proven effective in punishing sources who violate their negotiated traffic parameters while insulating behaving sources from troublesome effects[2]. Pulse shares this property[1], and is the means of enforcing adherence to negotiated DTW parameters in our two-level system. Pulse also guarantees fairness to sources in the sense that they receive bandwidth proportional to their peak rate[1]. We add no fairness enforcement to this, since DTW is designed to be used in the presence of such a queueing discipline.

Our reasoning behind using the queuing discipline to guarantee these aspects of performance is simple; queuing disciplines have already proven effective in controlling them. Additional meddling from the feedback scheme may well cause more problems than it solves. The whole purpose of a two level system is to let each level deal with those aspects that it is best suited to controlling, and in our system those aspects are best handled by the queueing discipline.

All the algorithms and enhancements mentioned here have been simulated, and the results of these simulations are discussed in detail in Section 5.

4.1. Previous Work

In formulating the DTW congestion control feedback system we took many cues from existing feedback congestion control strategies. In situations where one cannot receive help from the network, methods for estimating congestion and providing a feedback control in the face of delays under TCP were explored by Jacobson[9]. Jacobson estimates congestion by monitoring dropped packets and round trip delays, and uses that information to regulate the size of his packet windows.

The DECbit scheme[7],[8] provides both a robust practical congestion control strategy and a detailed analysis. DECbit relies on smart switches recognizing congestion and informing sources who adjust traditional packet windows to achieve a global throughput optimum. Although we do not use packet windows, many of their observations are useful in our environment.

4.2. Basic System

The goal of DTW is to allow bursty sources to receive good throughput while minimizing packet losses. No guarantees of service beyond a negotiated average throughput are made. For clarity we will describe the scheme assuming multiple sources sending through one switch to their destination. The study of multi-hop paths is work in progress.

Earlier work has shown that when sources are obeying the DTW admission control policy, the queue lengths at a switch are proportional to the size of the time window [1].

Because of this, we applied conventional feedback ideas to constructing the DTW feedback system, taking our cue from empirically successful packet window throughput control schemes such as DECbit [7] and TCP with Slow-Start[9].

Throughout this paper we use the terms *open* and *close* in the following way when applied to time windows. To open a MTW is to expand or increase it, allowing sources using that window to display more bursty behavior. To close a MTW is to reduce or decrease that time window, forcing sources to display less burstiness. Open and close are both relative terms; closing a window means reducing it, not setting it to its smallest value, likewise opening a window should not be construed to mean allowing sources their maximum burstiness.

The switch is the controlling entity in the DTW scheme. It maintains information about the size of the current MTW and is responsible for opening or closing it based on current measured traffic. Time windows are opened and closed using an additive increase/multiplicative decrease scheme, as described by Chiu and Jain [8]. This means that a constant is added to the size of the time window to open it, and that the window is multiplied by a number less than one to decrease it. The number added to the window is called the *additive increase unit,* and the number used to close the window is the *multiplicative decrease factor.* In the basic version of DTW, they are both unchanging parameters, but the multiplicative decrease factor is varied when damping (Section 4.5) is used.

Consider a switch with k sources transmitting to sinks on the other side of the switch. The switch maintains the MTW for the VCs running through it, and communicates changes to the sources. The sources maintain their time windows using the ratios negotiated at connection set up time. Each VC's time window is that VC's ratio multiplied by the current MTW ($I_k = I_{\max} r_k$) for VCs using the same switch. The initial $I_{\max}$ is communicated to the sources at connection set-up time, and changes are communicated from time to time as feedback.

The algorithm for adjusting time windows is as follows. The switch monitors its queue length starting from an arbitrary point in time. Define a monitoring interval to be a times the length of the current MTW, where a is an integer parameter of the switch determined at switch initialization. This means that monitoring intervals are dynamic. Because of this, when the time window is large, the network must be stable over a longer period for additional burstiness to be allowed.

If a monitoring interval elapses with no packet loss, the MTW is additively opened, and all VCs can open their windows. If packets are lost any time during a monitoring interval, the MTW is immediately closed, with sources closing their windows. Once the MTW is opened or closed, it is not opened or closed again until a round trip delay has passed, so that the system can settle into its new configuration before any new action is taken. It can take up to half a round trip time for the change to propagate to the source, and another half a round trip time before the traffic at the switch reflects the new value of MTW. Since the MTW is always greater than the maximum source to switch delay, this is mainly an issue for repeated closings of the window. However, when the MTW changes, the monitoring interval is immediately recalculated, and monitoring resumes. The algorithm is summarized in Table 4.1.

Since the switch is capable of injecting packets into the network, the process of opening and closing time windows is simply one of sending packets to the controlling entities at each source which then open or close their windows. Although we assume that no control messages are lost, correcting that is a simple matter of retransmitting packets. It may even be assumed that data packets have a header which includes that source's current opinion of the current MTW. This header could be used to insure that the whole system is in agreement about the current MTW.

One drawback of adjusting only the MTW is that all sources have their burstiness adjusted when the network changes state. Care must be taken to insure that when sources misbehave, the policing provided by the queueing discipline is sufficient to detect the difference between the network simply being in a state where it cannot support the aggregate burstiness of sources who are all sending within their negotiated parameters, and the situation where one or more sources are misbehaving. We do not deal with this issue directly in the simulations, but suggest some possible actions here.

Event	Action (taken at a switch)
A monitoring interval has elapsed without packet loss	Open the MTW, and restart queue length monitoring, using new MTW as the basis for the monitoring interval. Inform all sources of new MTW.
A packet is lost and more than a round trip time has passed since the last change to the window	Close the MTW, and restart the monitoring of queue length, using the new MTW as the basis for the monitoring interval. Inform all sources of the new MTW.
A packet is lost and less than a round trip time has passed since the last change to the window	Do nothing, let the system stabilize.

Table 4.1: The Adjustment Algorithm

The policing aspect of Pulse or another Generalized Virtual Clock discipline can detect packets sent in violation of negotiated parameters. The options for what to do with this traffic depend on how tolerant one wishes to be of misbehaving sources. The packets can be discarded systematically, tolerated if there is network slack, tolerated up to a threshold, etc. These choices are similar to the choices for red packets in the PlaNET network[5], although those packets are an attempt to statistically multiplex the network, and the packets under consideration in the DTW network indicate a misbehaving source. In any case, packets send in defiance of the negotiated parameters should not be considered by the feedback system, and their presence should not cause packet losses to well-behaved sources.

It has been argued that some sources cannot modulate their burstiness, or can modulate it, but must be guaranteed a certain amount of burstiness or their connection is useless. A way to provide such minimum burstiness to sources is to insure that there is a minimum value of the MTW that is strictly enforced. At connection set-up time, sources will be informed of the minimum MTW in the packet that acknowledges connection acceptance. If the source cannot operate under that burstiness constraint, it will have to drop the connection. Later work may allow the minimum MTW to be negotiated between source and switch, but as it is a function of the resource reservation policy, it is beyond the scope of this paper.

4.3. Adding and Deleting Sources

Part of what makes an oscillating feedback system attractive is that the oscillations make it responsive to changes in the system being monitored. This remains true under DTW, even though the feedback system is adjusting the burstiness rather than the throughput.

Consider a new source entering the system. It must naturally request a connection and negotiate time window parameters $(\lambda, \hat{\lambda}, r)$with the network. Once this is accomplished and the new source begins sending, it is likely that the burstiness of other sources should be curtailed in much the same way that a standard window-based congestion scheme would cut down the throughput of other sources. DTW detects this condition in the same way as a traditional packet window-based system, by the queue length growing beyond the threshold for packet loss, and reacts by closing the MTW, perhaps more than once in succession. Once the burstiness of all the sources has been reduced to a level where the switch queue is safe from loss at the switch the oscillations return and a new stable point is reached.

A source leaving the network causes the reverse effect. Once a source leaves, the MTW does not close at the same point in the upward oscillation, and will soon converge to a new stable point.

Strictly speaking the DTW feedback system can handle sources entering and leaving the system without modification, a fact that has been verified in simulation. However, since DTW operates in an environment that requires a new source to establish a VC, we can take advantage of the prior notification to anticipate a new source entering the network. For example, on receipt of a VC set-up message the switch can immediately close the MTW, lessening the chance that packets will be lost during detection of the new source. This has proven to be a useful performance enhancement.

Simulation shows that this simple feedback method is quite effective in controlling the effects of bursty traffic on queue length. It is also important to note that unlike traditional packet window-based schemes that modulate throughput to regulate queue lengths, DTW regulates only the variance of the streams. The throughput of the sources remains unchanged as the time windows change size. Still, some performance enhancements can be made, and the remainder of this section will discuss them.

4.4. Loss Thresholding

It is not necessary to wait until a packet is lost to begin closing the MTW. One can easily define a high water mark below the capacity of the switch, and close the window when the switch queue length exceeds it. This simple optimization greatly reduces loss without adversely affecting throughput. The high water mark is termed a *loss threshold* in this paper.

Loss thresholding affects other performance metrics than packet loss. It establishes a target value for the switch queue length, and so can be used to adjust other system parameters that depend on the expected queue length. An example of this is reducing the per packet expected delay by lowering the loss threshold value. When deciding on a loss threshold value one must weigh performance requirements, such as end-to-end delay and packet loss probability, against network utilization factors, such as switch buffer utilization and the number of connections that the network can accept.

4.5. Damping Oscillations

Feedback systems, especially ones using additive increase/multiplicative decrease adjustment algorithms, lead to oscillations of the window size about a fixed stable point[7],[9],[13]. The DTW scheme is no exception, with the queue length and MTW size both increasing until the loss threshold is reached, decreasing as the MTW closes, and then oscillating upward again.

Although throughput does not suffer from this oscillation as it would in a traditional window-based congestion control, it is desirable for these oscillations to be damped, so that the source behavior can move toward a steady state when the system is in one. However, we must also be wary that damping the oscillations does not render the system unable to respond to changes in the network state, such as addition of a new source, or deletion of a source.

The goal of our damping is to detect cases where the MTW is repeatedly opening to the same size and then being closed as the queue length threshold is exceeded. When this is detected we want to reduce the amount we close the MTW, so the window can stay closer to the stable point. When a change in system configuration occurs, we want to close the MTW by a larger margin so that the system can seek a new stable point.

Define two constants $MDF_{\min}$ and $MDF_{\max}$ as numbers between 0 and 1. Let $MDF_{\min}$ be the multiplicative decrease factor used to close the MTW when the system is in a stable state. $MDF_{\min}$ will produce the minimum change in the MTW when used to close it. On the other hand $MDF_{\max}$ is the multiplicative decrease factor used to close the MTW when we are unsure of the system stability. It allows a greater range of oscillation, allowing DTW to seek a stable point more quickly. The damping algorithm adjusts the current multiplicative decrease factor (*MDF*) between these two extremes depending on how stable the measured switch queue lengths have been. Notice that the max and min subscripts refer to their effects on the MTW not their values; in fact, $MDF_{\min} > MDF_{\max}$.

The following algorithm implements this idea. *MDF* is initially set to $MDF_{\max}$ since we are unsure of the system state. Every time the switch decides to close the window it saves the current value of the MTW, which must be the highest value that the MTW took on since the last time it was closed, and closes the window using *MDF* as the multiplicative decrease factor. Unless the network is on the verge of collapse, the MTW will eventually begin to increase again, until it again reaches a point where the switch queue length becomes too full and it again decides to close the window. It first checks to see if the MTW is within one additive increase unit of the saved MTW value. If the two values are within one additive increase unit, it is taken as evidence that the system is in a stable state, so *MDF* is adjusted to be closer to $MDF_{\min}$. Our method of arriving at a new *MDF* is to add half the difference between *MDF* and 1 to *MDF*, stopping at $MDF_{\min}$. ($MDF = \min\left(MDF + \frac{(1-MDF)}{2}, MDF_{\min}\right)$). If the two MTW values are not within one additive increase unit we assume that some change to the system's status quo has occurred, for example a new source has established a connection, and reset *MDF* to $MDF_{\max}$. Care must be taken in the choice of $MDF_{\min}$ to insure that in its most stable state it is still possible for the system to react to changes in the network state.

Simulation shows that this leads to a system that exhibits small oscillations in steady state, and still is able to respond to changes in the network state.

4.6. Delay and Feedback

In Section 3 we discussed how the admission control strategy of DTW impacts the effect of control delay. Due to the natural congestion clearing effects of the DTW admission control, the feedback component of the congestion control is free to take a more preventive stand than in most congestion control schemes, which results in the simple but effective algorithms discussed above. This was borne out by simulations, which were very insensitive to the control delay.

5. Simulation Results

This Section discusses the simulation results in detail. We begin by describing the simulation configuration and parameters, and then discuss individual simulations that illustrate specific features of DTW.

5.1. Simulation Parameters and Configuration

In all cases, multiple sources are sending through a switch to each other. The sources are all modeled as packet train sources [10]. They send trains of packets where the start of each train is exponentially distributed and the number of packets in the train is uniformly distributed. While sending a train, each packet is sent as close to its predecessor as possible, and all packets are sent at the source's peak rate. No transport level protocol is modeled beyond keeping the sources operating in accordance with DTW rules, so no acknowledgements or retransmissions are modeled.

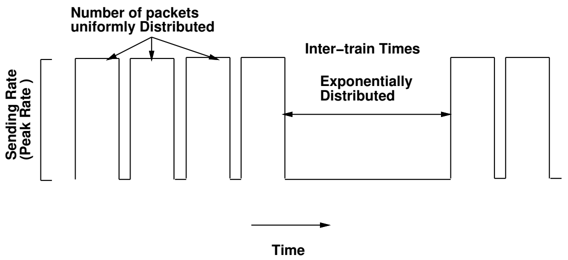

Figure 5.1: A Packet Train

The negotiated parameters for each source are $\bar{\lambda} = 50$ Mbits/sec, $\hat{\lambda} = 100$ Mbits/sec and r varies between 1.0 and 0.2 on a source by source basis. The switch service rate (μ) is 1 Gigabit/sec, and there are 20 sources in each simulation. Except for the delay study, the line propagation delays are kept

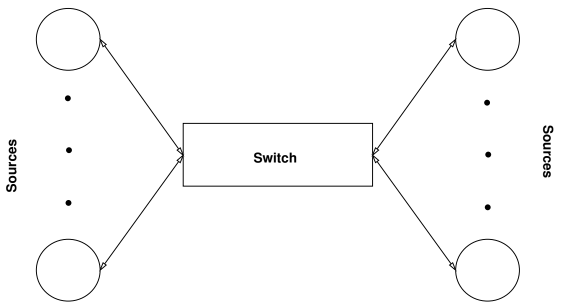

Figure 5.2: Simulation Configuration

at 10 msec. The switch has 100 Megabits (12.5 Megabytes) of buffering. Line capacities are presumed high enough to handle all traffic. Packet size is 1000 bytes or 8000 bits. In all simulations the switches use an additive increase unit of 10 msec, and a base multiplicative decrease factor of 0.5. The initial value of I_{max} was 1.0 sec, and was varied throughout the simulation as per the algorithms given in Section 4.

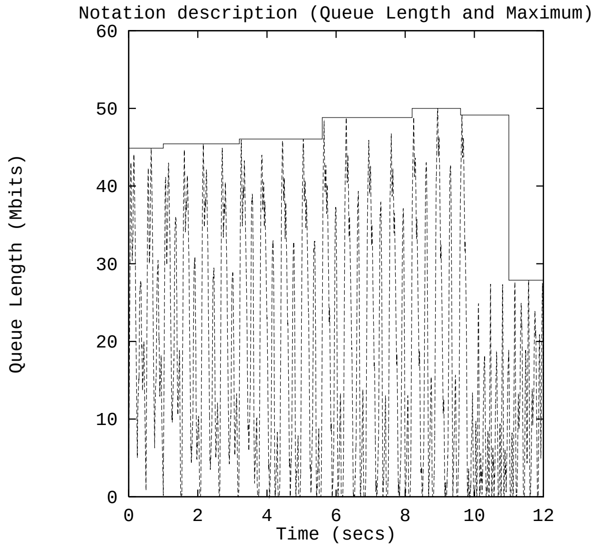

Figure 5.3a: Notational example

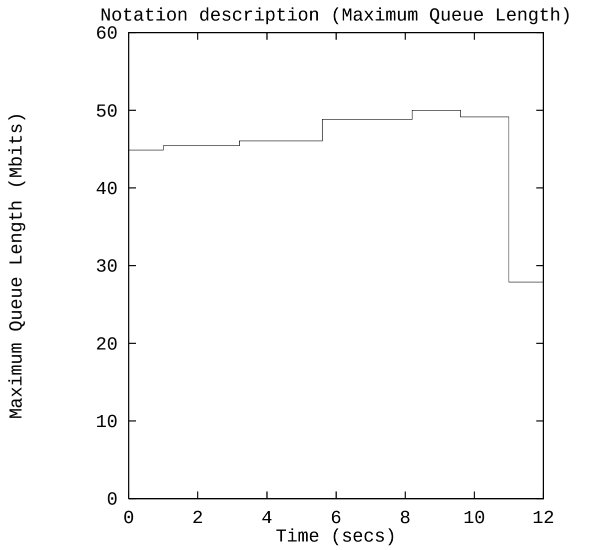

Figure 5.3b: Notational example

5.2. Notation of Graphs

In the graphs that follow, the convention is followed of representing each measurement interval by a line segment whose horizontal length is the length of the monitoring interval, and whose height indicates the maximum queue length at the switch during that interval. This gives a bounding box of the queue length. The following graph shows an example, as well as showing the actual queue lengths used to determine the bounding box. The box is solid, and the actual queue length is a dashed line. Figure 5.3a shows both the maximum queue length for the monitoring intervals, and the actual measured queue length used to define it. Figure 5.3b shows only the maximum queue length, the convention that will be followed in all other graphs presented here; *i.e,* later graphs do not show actual queue lengths, but rather the maximum queue length in monitoring intervals. Also of interest in the figure is the spiked nature of the queue length graph. This is due to the constant renewals of the queue lengths caused by DTW admission control, as described in Section 3.3. Since these renewals are so frequent, the bounding box notation is used to make the graphs legible. Also the repeating patterns of high and low spikes are caused by resonances in the time windows of different sources. All the time windows are integral divisors of the MTW to insure that the renewals occur, and as such they also form harmonics, since they are somewhat synchronized by each change to the MTW.

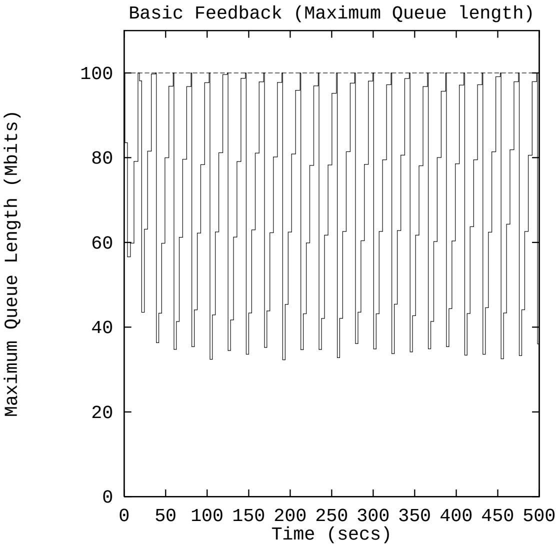

Figure 5.4a: Basic DTW Simulation Queue Length

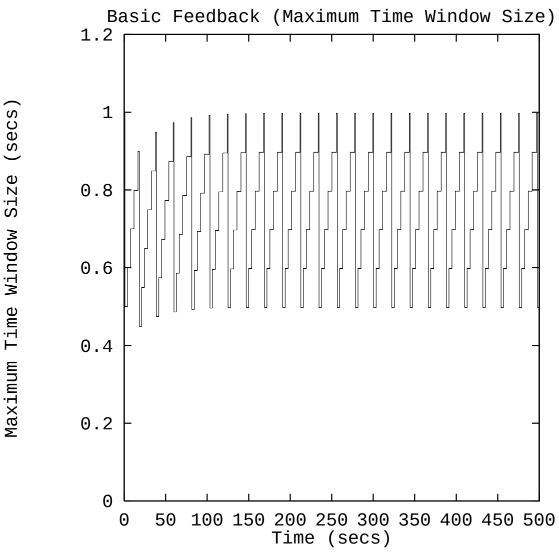

Figure 5.4b: Basic DTW Simulation Time Window Size

5.3. Basic DTW Simulation

The basic feedback algorithm discussed in 4.2 is simulated in this subsection. There are 20 sources each sending in accordance with the DTW rules into the switch with their parameters as above. Figure 5.4 summarizes the switch queue length and MTW changes over time. The queue length graph, like all queue length graphs that follow is in the notation of Section 5.2. The horizontal dashed line indicates the buffer capacity of the switch.

This simulation run used a monitoring interval of 5 times the MTW ($5I_{\max}$), that is, the value of a was 5. The simulation showed a throughput of 994.8 Mbits/sec, switch utilization of 99.5%, and packet losses were 0.067%. All throughputs in this section are aggregate throughputs of all VCs passing through the switch. Also throughput is measured by packets that leave the switch, so lost packets are not counted in switch throughput. Smaller monitoring intervals tend to lose more packets by opening the MTW too quickly. Some typical results are for $2I_{\max}$, $5I_{\max}$, and $10I_{\max}$ are given in table 5.1. Notice that the maximum queue length at the switch closely tracks the MTW, as expected, and that the system quickly settles into a pattern of oscillations about a stable point. Losses occur because the system waits for packet loss to shut down the MTW.

Monitoring Interval	Throughput (Mbits/sec)	Loss (percent)	Switch Utilization
2	990.6	0.067	99.1%
5	994.8	0.042	99.5%
10	997.2	0.008	99.7%

Table 5.1 : Basic DTW Simulation Summary

5.4. Adding and Deleting Sources

These simulations show the effect of adding and deleting a source to a stable network. All these simulations used a threshold of 90% of available buffers in the switch, see Sections 4.4 and 5.5 for more details on the use of the loss threshold. In all cases, the change in the system (*i.e.,* the addition or deletion of the source) occurred at t = 200 seconds, which is marked with a vertical line in all graphs.

First we present typical results for deleting a source in Figure 5.5. All sources are sending at the rates described in 5.1, and a source with a ratio of 0.50 stops sending at t = 200. The maximum number of buffers in the switch is the dashed line at 100 Mbits, and the loss threshold is 90 Mbits of buffer space shown by the dotted line. The monitoring interval is $5I_{\max}$.

Notice that when the source disappears, the system is less stressed, and the queue length graph no longer peaks as sharply. After the source disappears, the MTW oscillates about a larger average value, as our intuition says it should.

In considering addition of new sources, we present two simulations. The first is simply adding a source and allowing the basic DTW scheme to restabilize the system, and is shown in Figure 5.6. These graphs are from a simulation where a source with ratio 0.50 starts sending at t = 200. All sources have the same rates as before, loss thresholding is used, and the monitoring interval is $5I_{\max}$.

In the graphs of the addition simulation, we see the opposite of the deletion graphs. Queue lengths go from peaking less sharply to peaking more sharply after the new source appears, and the MTW stabilizes at a considerably smaller value.

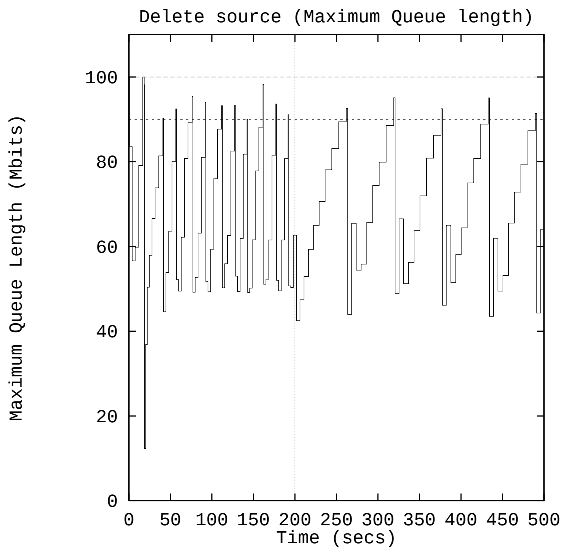

Figure 5.5a: Deleting a Source (Queue Length)

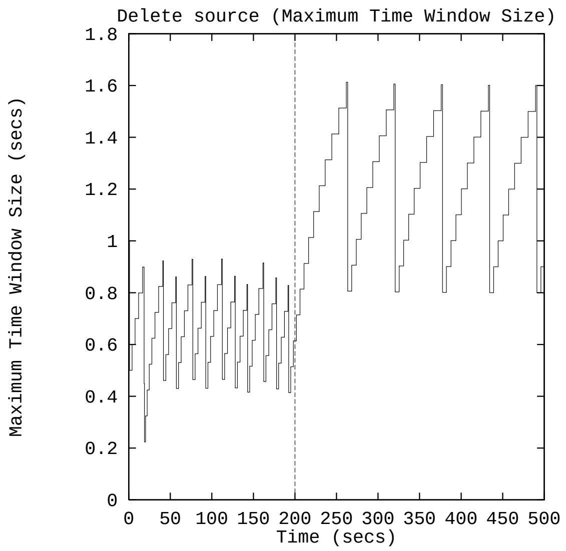

Figure 5.5b: Deleting a Source (Time Window Size)

The second simulation, shown in Figure 5.7, takes advantage of the VC set-up to anticipate a new VC. When it receives the set up message, it preemptively closes its MTW. All parameters are the same as the previous simulation.

With anticipation, when the new source enters the system, there is not as high an peak, and the system can more smoothly return to a steady state. Notice that in Figure 5.6 packets were lost when the source was added at *t*=200, but in Figure 5.7, the anticipation prevents such a loss. Both simulations had the same sequence of packet arrivals.

5.5. Adjusting the Loss Threshold

In these simulations we examined the use of the loss threshold or high water mark. These simulations use the same scenario as the basic simulations in Section 5.3, adjusting the high water mark to 90% and keeping all other parameters equal. Figure 5.8 shows the graphs obtained from that simulation. The additional dashed line is the loss threshold.

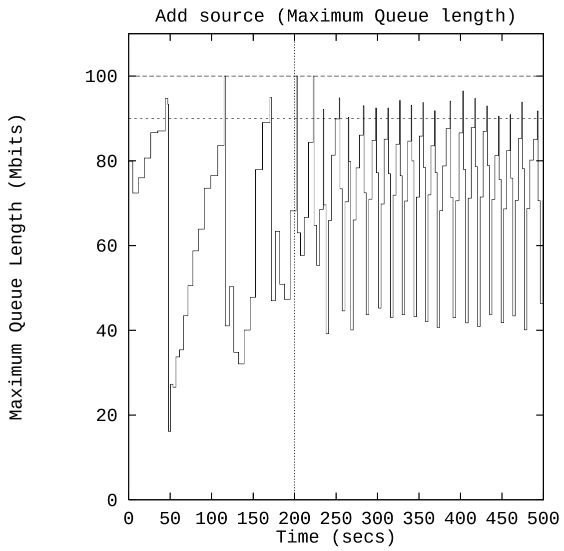

Figure 5.6a: Adding a Source (Queue Length)

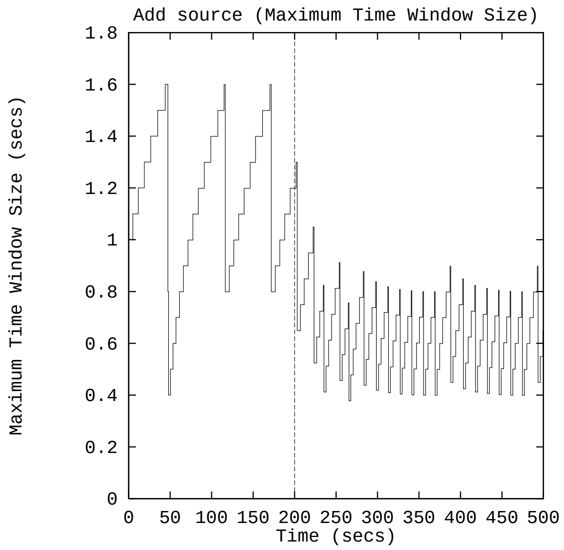

Figure 5.6b: Adding a Source (Time Window Size)

For comparison purposes here, Table 5.2 contains the simulation results for the same three typical runs given in Table 5.1.

Thresholding reduces losses while maintaining throughput. The MTW remains smaller, meaning that traffic is required to be less bursty, but as long as the sources can operate within that constraint, thresholding is useful in reducing or eliminating losses.

In these simulations, the threshold was set to 90% of available buffers full. This means that the queue lengths at the switch were allowed to grow rather large before the network took action to reduce them. Basically we were using the buffering at the switch to allow very bursty traffic, and so there were relatively full queues, and occasional loss. Another choice would be to move the threshold much lower, with the intention to reduce delay while allowing less switch buffering to be used, thereby reducing source burstiness.

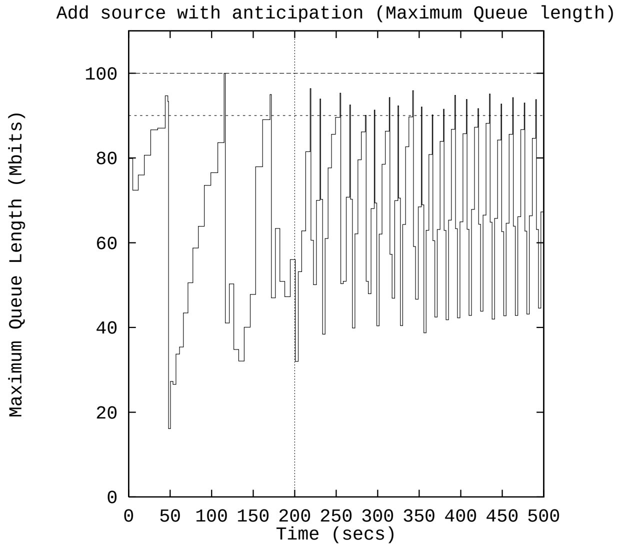

Figure 5.7a: Adding a Source with Anticipation (Queue Length)

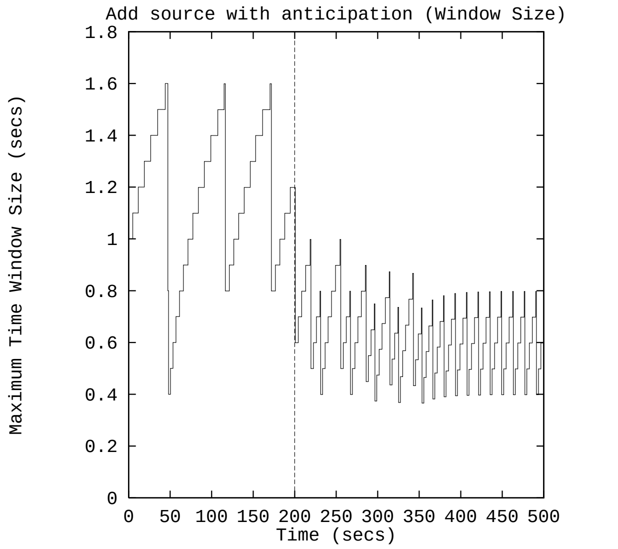

Figure 5.7b: Adding a Source with Anticipation (Time Window Size)

5.6. Damping

In these simulations, the damping algorithm discussed in Section 4.5 was used. This results in the MTW settling into a tighter oscillation, thus sources did not need to change the modulation of their burstiness by a wide margin. The graphs in Figure 5.9 show the damping of the oscillations in both queue length and MTW size. The monitoring interval is $10I_{\max}$. The upper horizontal line is the buffer size of the switch, and the lower is the loss threshold. $MDF_{\max}$ was 0.5 and $MDF_{\min}$ was 0.9.

The reader will notice that the peaks of both the MTW and the queue length oscillations are not constant. This is due to the fact that the multiplicative decrease does not necessarily place the low point of the oscillation an integral number of additive increase units away from the same peak that closed it. Since the MTW only opens by whole additive increase units, it closes at a slightly different MTW value than the previous iteration. This is why the algorithm damps the oscillation if the two peaks are within one additive increase unit.

It is difficult to sum up the results of these simulations numerically. Essentially the goal of this enhancement is to make the MTW size as stable as possible while the system is in a steady state. From the graphs, one can see that this criterion is met, but it is difficult to quantify. We can show, however that stability does not sacrifice other advantages of the DTW, as seen in Table 5.3. Note that throughput remains high and losses remain low.

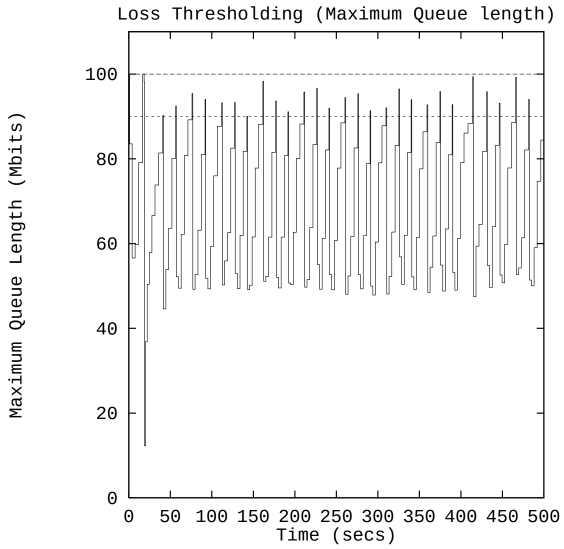

Figure 5.8a: Low Threshold Simulation Queue Length

5.7. Delay and DTW

These simulations compare DTW to a packet window-based congestion control. The packet window control uses a standard packet window whose size represents the number of packets it is permitted to have outstanding. If it wishes to send a packet and the number of currently outstanding packets is less than the current window size, it sends the packet, otherwise it waits for an acknowledgement and then sends the packet. Each source reacts to congestion using the same additive increase/multiplicative decrease algorithm that DTW uses: if a round trip time passes without a source losing a packet, it increases its window by 10, if it has lost a packet that interval it reduces it by a multiplicative decrease factor. The system was tested using multiplicative decrease factors of 0.5 and 0.875. The window opens additively by a comparatively large additive increase unit because the packet window-based sources must send enough data to utilize the Gigabit/sec switch, and lesser values cannot reopen a window of the size needed in any reasonable time. Sources have perfect knowledge of all packets lost in a round trip time, and receive that knowledge every elapsed round trip time. The initial window size is 5000 packets (5 MBytes). So the twenty sources can have 800 Mbits in flight at t=0. This compensates somewhat for the lack of a slow start[9] for these sources.

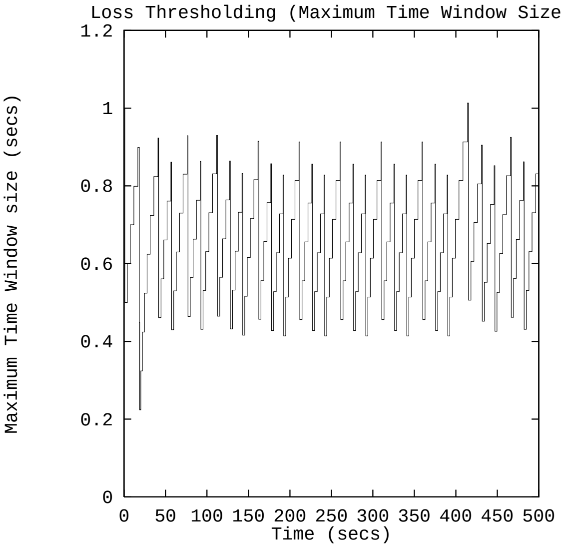

Figure 5.8b: Low Threshold Simulation Time Window Size

Monitoring Interval	Throughput (Mbits/sec)	Loss (percent)	Switch Utilization
2	990.3	0.007	99.0%
5	994.6	0.003	99.5%
10	996.7	0.001	99.7%

Table 5.2 : Loss Thresholding Simulation Summary

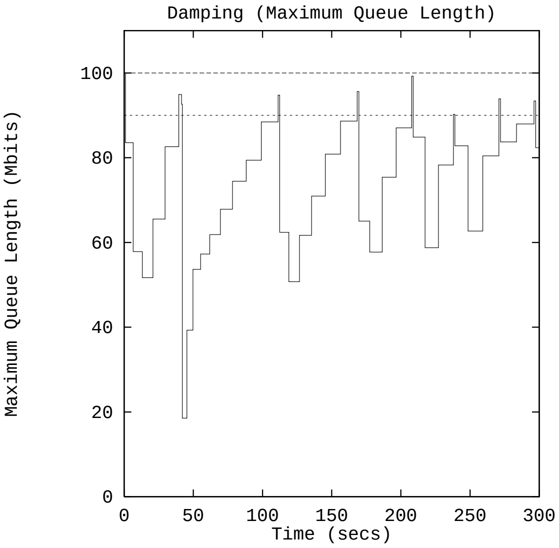

Figure 5.9a: Damping Simulation Queue Length

Results were tabulated for 500 second runs under DTW using the same sources as before under several different delays. The packet window simulations used the same arrival patterns,

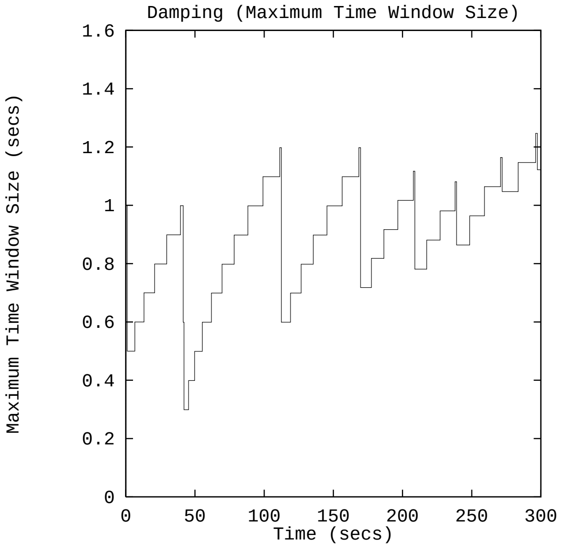

Figure 5.9b: Damping Simulation Time Window Size

Monitoring Interval	Throughput (Mbits/sec)	Loss (percent)	Switch Utilization
2	992.2	0.016	99.2%
5	995.3	0.009	99.5%
10	997.3	0.002	99.7%

Table 5.3 : Damping Simulation Summary

but regulated the sources using the generic packet window scheme described above. The results are summarized below. *MDF* indicates the multiplicative decrease factor used in the packet window simulations, the DTW simulations all used an *MDF* of 0.5.

DTW control			
Switch to Source Delay	Throughput (Mbits/sec)	Loss (percent)	Switch Utilization
10 msec	994.6	0.003	99.5%
50 msec	994.9	0.001	99.5%
100 msec	994.8	0.002	99.5%
250 msec	995.0	0.002	99.5%
500 msec	995.8	0.002	99.6%
750 msec	997.6	0.004	99.8%
900 msec	998.0	0.016	99.8%

Table 5.4 : DTW Delay Simulation Summary

Notice that the packet windowed system performs well at low delay, but breaks down seriously as the delay increases. Granted, the losses as a percentage of data sent never get very high, but when you combine the rising losses with the drop in throughput, the performance is unacceptable. DTW, on the other hand, remains stable across the range of delays. In fact throughput rises slightly with increasing delay. This is an artifact caused by the fact that the MTW cannot be less than the source to switch delay. High source to switch delays put an excessively high lower bound on the MTW, allowing more burstiness, but also causing more loss.

Packet window control MDF=0.5			
Switch to Source Delay	Throughput (Mbits/sec)	Loss (percent)	Switch Utilization
10 msec	999.9	0.168	99.9%
50 msec	999.9	0.149	99.9%
100 msec	930.9	0.197	93.2%
250 msec	594.1	0.353	60.1%
500 msec	437.9	0.472	45.5%
750 msec	307.6	0.672	31.3%
900 msec	466.2	0.440	48.7%

Table 5.5 : Packet Window Delay Simulation Summary

Packet window control MDF=0.875			
Switch to Source Delay	Throughput (Mbits/sec)	Loss (percent)	Switch Utilization
10 msec	999.9	0.728	99.9%
50 msec	999.9	0.630	99.9%
100 msec	772.3	0.959	77.2%
250 msec	432.5	1.697	43.2%
500 msec	263.0	2.762	26.3%
750 msec	199.7	3.579	20.0%
900 msec	191.5	3.718	19.0%

Table 5.6 : Packet Window Delay Simulation Summary

6. Future Work

There are many directions that the exploration of DTW can take. One of these is continued simulation of more complex systems: more switches, different source behavior models multi-hop VCs. It will also be interesting to see what notions of quality of service can be introduced to DTW to deal with data sources that are less tolerant of delay and more so of loss.

Also important is more closely linking DTW and the Generalized Virtual Clock queueing discipline that it is designed to complement. Right now the two are designed to co-exist, but not to really interoperate. The closer the links between the two become, the tighter a hold we feel that we will have on congestion control.

Also, as programmable smart switches become available, the chance to do an actual implementation becomes increasingly attractive. We will use the XUNET switch of the BLANCA Gigabit testbed to experiment with DTW/GVC. Simulation results are encouraging, but it would be far more satisfying to test our ideas on real traffic.

It would also useful to obtain a general mathematical model for the behavior of sources under DTW. Such a model would aid in capacity planning and call-time allocation for Gigabit networks running DTW. It would also prove useful in determining the performance of DTW on traffic that behaves differently from our assumptions.

7. Conclusions

We have presented a feedback congestion control scheme, DTW, for use in high speed wide area networks (HSWANs) based on controlling sources' variance. The scheme is very general and robust, and requires only that sources submit to a simple traffic constraint. Throughput is not modulated by the scheme. The scheme has been simulated for the single switch case, and is shown to be sound.

Aspects of the admission control have been found that bound the congestion time of switches using DTW. This property has been verified by both analysis and simulation, and has been shown to mitigate the effects of the high bandwidth-delay product characteristic of HSWANS.

The DTW feedback has also been shown to be very extensible, and we have demonstrated several extensions that improve the system's performance. These include loss thresholding, oscillation damping, and anticipation of a new source. All of these favorably change the performance of DTW without sacrificing its high throughput.

We have identified the next step for the DTW system and are confident that it can be more tightly tied to the Generalized Virtual Clock queueing discipline. As new virtual clocks arise it will be interesting to adapt DTW to take advantage of their specific performance characteristics. Other areas include simulating the scheme across multiple switches and with a wider variety of traffic types.

References

[1] Mukherjee, A., L. Landweber, T. Faber, ''Dynamic Time Windows and Generalized Virtual Clock: Combined Closed Loop/Open Loop Congestion Control,'' *Proc. IEEE INFOCOM, May 6-8, Florence, Italy,* May 1992.

[2] Zhang, L. ''Virtual Clock: A News Traffic Control Algorithm for Packet Switching Networks'' *Proc. ACM SIGCOMM, Philadelphia,* 1990.

[3] Fraser, A.G., C.R. Kalmanek, A.E. Kaplan, W.T. Marshall, R.C. Restrick, ''XUNET 2: A Nationwide Testbed in High-Speed Networking,'' *Proc. IEEE INFOCOM, May 6-8, Florence, Italy,* May 1992.

[4] Turner, J., ''New directions in communications (or which way to the information age?),'' *IEEE Comm. Magazine,* vol. 24, pp. 8-15, Oct. 1986.

[5] Bala, K., I. Cidon and K. Sohraby, ''Congestion Control in High Speed Packet Switched Networks,'' *Proc. IEEE INFOCOM, San Francisco,* pp. 520-526, June 1990.

[6] Sidi, M., W. Liu, I. Cidon and I. Gopal, ''Congestion control through input rate regulation,'' *Proc. IEEE GLOBECOM, Dallas,* pp. 1764-1768, Nov. 1989.

[7] Ramakrishnan, K. K., R. Jain, ''A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer,'' *Proc. ACM SIGCOMM, Stanford,* pp 303-313, 1988.

[8] Chiu, Dah-Ming, and R. Jain, ''Congestion Avoidance in Computer Networks With a Connectionless Network Layer Part III: Analysis of the Increase and Decrease Algorithms,'' *Technical Report DEC-TR-509, Digital Equipment Corporation,* 1987.

[9] Jacobson, Van, ''Congestion Avoidance and Control,'' *Proc. ACM SIGCOMM, Stanford,* pp 314-329, 1988.

[10] R. Jain and Routhier, S., ''Packet Trains - Measurement and a New Model for Computer Network Traffic'' *IEEE Journal on Selected Areas in Communications,* SAC-4(6):986-995, September 1986.

[11] Ramamurthy, G., and R.S. Dighe, "Distributed source control: a network access control for integrated broadband packet networks," *IEEE J. Select. Areas Commun.,* vol. 9, pp. 990-1002, Sept. 1991.

[12] Golestani, S. J., "A framing strategy for congestion management," *IEEE J. Select. Areas Commun.,* vol. 9. pp. 1064-1077. Sept. 1991.

[13] Bolot, J., and A. U. Shankar, ''Dynamical behavior of rate-based flow control mechanism,'' *ACM SIGCOMM Computer Communication Review, vol. 20, pp. 35-49,* April 1990.