ASYNCHRONOUS DISTRIBUTED
FLOW CONTROL ALGORITHMS

by

Jeannine Mosely

B.A. University of Illinois, Urbana IL (1974)

B.S. University of Illinois, Urbana IL (1977)

S.M. Massachusetts Institute of Technology (1979)

E.E. Massachusetts Institute of Technology (1980)

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1984

Signature of Author Jeannine Mosely
Department of Elec. Eng. and Comp. Science
28 May 1984

Certified by________________________________
Prof. Pierre A. Humblet, Thesis Supervisor

Accepted by________________________________
Prof. Joel Moses, Cnairman Dept. of E.E.C.S.

ASYNCHRONOUS DISTRIBUTED FLOW CONTROL ALGORITHMS

by

Jeannine Mosely

Submitted to the Department of Electrical Engineering and Computer Science, 29 May 1984 in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the Massachusetts Institute of Technology

Abstract

We consider algorithms for flow control in computer networks with fixed routing. The goal is to establish input rates, for each source-destination pair, that satisfy a particular fairness criterion. We describe several algorithms in which the input rates are calculated based on controls established by the links of the network. These controls are updated iteratively, using feedback information from the network. We show that the rates thus calculated converge to the desired values when the links are assumed to update synchronously, and without feedback delay. A model for asynchronous operation with delay is given, and we demonstrate for this model that the input rates calculated by the synchronous algorithms may fail to converge. We show how to modify the algorithms, by the introduction of an update protocol and by using more of the available feedback information, so that convergence of the rates is guaranteed.

We extend the model for asynchronous computation developed by Bertsekas [14] to get some results relating to general asynchronous distributed algorithms with update protocols. These results are used to give an alternate proof of the correct operation of one of the flow control algorithms.

We develop a computer program to simulate the flow control algorithms for a voice packet network. The simulation results indicate that the algorithms behave as expected for a network with static loads. However, when input loads change in imitation of real conversations, the control algorithms do not adapt fast enough to control the flows effectively.

Thesis supervisor: Prof. Pierre A. Humblet Title: Associate Professor of Electrical Engineering

Table of Contents

List of Figures

Table of Tables

Acknowledgements

I would like to thank Prof. Pierre Humblet, my thesis supervisor for all the assistance he has provided during the course of this work, and for bringing this problem to my attention in the first place. I would also like to thank my thesis readers, Prof. Bertsekas and Prof. Gallager, for all their attention, and especially their promptness when the deadline approached. I am especially grateful to Prof. Bertsekas for his suggestions leading to the results in Chapter 4. Prof. Gallager I thank, not only for his careful reading, which uncovered many errors that must otherwise have gone unnoticed, but also for his patience and continued support over the years.

To Allan Wechsler, my friend and help-mate, go special thanks: for assisting me with the simulation program in Chapter 5, for listening to my nonsense at all hours of the day and night, and for believing in me.

I would like to thank my advisor Prof. Al Drake, whose advice, encouragement and friendship have been invaluable.

I would also like to thank Symbolics, Inc., on whose machines the simulation was developed, and the M.I.T. Artificial Intelligence Lab, where the simulation was run. Thanks are due to the hackers and wizards of OZ for help rendered in generating the graphical output from the simulation data. Special thanks go to MARG, GUMBY and DLW of the "Dover Express", who arranged for the graphs to be delivered from Stanford at the last minute when M.I.T.'s own Dover broke.

I would also like to thank my Dad, who has encouraged me throughout my life to pursue my technical interests.

Chapter 1

Introduction

1.1 Background

Advances in packet switching techniques make packet switching a cost effective method for handling sporadic or bursty comminications traffic. The sporadic nature of voice and the desire to integrate voice and data in computer communication networks [1], [2], makes the idea of packet voice attractive. In this thesis, we consider the problem of limiting traffic flow in such integrated networks.

In a traditional circuit switched voice communication network, a given 2-way conversation is allotted two dedicated channels. But usually a user spends 50% of his time listening. In addition, pauses between words and phrases in the speech of the active user represent a source of wasted channel resources. These smaller units of uninterrupted voice are called "talk-spurts". The random nature of talk-spurts has long been exploited by the Bell System in their TASI algorithm, used on intercontinental lines [3]. Digital variations on TASI include Digital Speech Interpolation and Speech Predictive Encoding [4], [5].

In an integrated voice and data communication network, it is necessary to adopt flow control measures to limit the amount of information entering the network, and prevent congestion. While flow control techniques for data-only networks have reached a high level of sophistication [6], little is known about flow control for voice. The different delivery requirements of voice and data demand a different approach to the problem of flow control for each. While considerable delay may be acceptable in a data packet, the same delay would cause a voice packet to be discarded by the receiver as "too late". Conversely, voice may suffer considerable degradation due to errors and still be intelligible, while the same errors in a data packet make it worthless.

Traditional methods of flow control for voice simply block the initiation of new calls. TASI type systems may even block new talk-spurts, resulting in clipping of the received signal. Loss of more than about .5% of the signal by clipping has proved unacceptable.

The idea of embedded coding, first proposed at the Naval Research Laboratory [7], provides a new approach to voice flow control [8]. In embedded coding, speech is encoded into priority ranked packets. The lower priority packets can be discarded as needed, while the remaining packets still provide an intelligible, though degraded, signal. Hence, we have traded clipping for distortion. The level of network congestion that results in unacceptable clipping for a call blocking scheme is much lower than that required to render embedded coding speech unintelligible.

Low priority packets can be discarded at their point of entry into the network, as well as at the point of congestion, resulting in a variable rate encoding scheme. Clearly, it is better to discard entering packets when possible, to prevent unnecessary waste of network resources.

An algorithm for voice flow control using embedded coding has been studied by a group at Lincoln Laboratories, using a computer simulation [8]. In their model, conversations are conducted over fixed routes. Low priority packets are discarded at congested nodes, and terminals report the received rates to senders, which reduce their input rates accordingly. Provisions are included for allowing the senders to increase their rates when the network is lightly loaded.

The primary objectives of this scheme are to maintain stable operation of the network, while preventing excessive delays due to congestion and providing the highest level of service possible to each user.

This last criterion gives rise to the question of how to allocate network resources in a "fair" manner, while giving everyone the best possible service. Hayden [9] and Jaffe [10] simultaneously and independently arrived at a concept of "fair" rate allocation, which was generalized by Gafni and Bertsekas [11], [12], and is defined in the next section.

1.2 Problem Model

We will use the following model to study the problem of voice flow control. Consider a network $\mathcal{N}$ which consists of a set of links $\mathcal{L}$ and sessions $\mathcal{S}$, where a session is a source-destination pair between which a conversation is taking place. Each link j has an associated capacity c_j, where $c_j \geq 0$. Each session is assigned a path through the network, which is fixed for the duration of the conversation. We denote by $\mathcal{L}_i$, the set of links in the path of session i. We denote by $\mathcal{S}_j$, the set of sessions whose path contains link j.

Let $r_i(t)$ be the input rate of session i at time t. For now, we assume instantaneous propogation of data through the network, so that the component of flow on link j at time t, due to session i, is $r_i(t)$. (This assumption will be dropped later, in favor of a more realistic model.) Then we define the flow on link j at time t as

$$f_j(t) = \sum_{i \in \mathcal{S}_j} r_i(t). \tag{1.1}$$

We control the flow only by limiting inputs rates, and not by discarding packets when the links are congested. We wish to control the input rate for each session i and the flow on each link j so that the steady state rate $r_i = \lim_{t\to\infty} r_i(t)$ and steady state flow $f_j = \lim_{t\to\infty} f_j(t)$ exist, and satisfy the constraints for a fair allocation, as outlined below.

We would like for a fair rate allocation to be indifferent to the geographical separation of the source-destination pair. While priorities may be established for certain sessions, it should not be on the basis of distance. Furthermore, two sessions of the same priority should be assigned the same rate, if the rate of one can be traded for the other, without reducing the rate of any other session or violating other system constraints. This will make the network transparent to the session, in the sense that he cannot tell the length of the assigned path by the assigned rate.

We also require that each user be assigned the highest possible rate, while guaranteeing that the steady state flow on each link does not exceed a given function of the link capacity.

With this motivation, we give the definition of a fair allocation, first presented by Gafni [11]. First we need the following two definitions.

A vector $x=(x_1,\ldots,x_n)$ is said to be lexicographically less than or equal to $y=(y_1,...,y_n)$ if $x_i>y_i$ implies the existence of $j<i$ such that $x_j<y_j$. We write this as $x \underset{lex}{\leq} y$.

Given a vector $x \in R^n$, let $\tilde{x}$ denote a vector whose coordinates are some permutation of the coordinates of x. If the coordinates of $\tilde{x}$ have the property that $x_1 \leq x_2 \leq ... \leq x_n$, we call $\tilde{x}$ the <u>increasing permutation</u> of x, and we denote this vector by $\overline{x}$. (Note that $\overline{x} \underset{lex}{\leq} \tilde{x}$, for any permutation $\tilde{x}$ of x.)

<u>Definition</u>. Let X be any subset of R^n. We say that $x \in X$ is a <u>fair allocation</u> over X, if for all $y \in X$, $\overline{y} \underset{lex}{\leq} \overline{x}$.

We may think of X as a "feasible" set. The fair allocation vector solves the following set of nested problems. The first problem is to find a subset X_1 of X, such that the minimum coordinate of a vector $x \in X_1$ is greater than or equal to the minimum coordinate of any vector in X. Next we find a subset X_2 of X_1, such that the second smallest coordinate of $x \in X_2$ is maximized over X_1, and so on.

1.3 <u>Previous Work</u>

Hayden [9] gives a distributed algorithm which produces a rate vector $r=(...,r_i,...)$ that is a fair allocation over the set defined by

$$f_j < a_j c_j \qquad \forall j \in \mathcal{L}, \tag{1.2}$$

where a is some constant, $0<a\leq 1$. The rationale behind (1.2)

is simple: we restrict the steady state flow on each link to some fixed fraction of link capacity, reserving the unused capacity as a buffer against transient fluctuations in flow. We call $a_j c_j$ the effective capacity, and henceforth, when we refer to a link's capacity it is assumed that we mean the effective capacity.

Jaffe [10] gives an algorithm such that the vector $(\ldots,b_i r_i,\ldots)$ is a fair allocation over the set defined by

$$b_i r_i \leq c_j - f_j \qquad \forall i \in \mathcal{S}, \quad \forall j \in \mathcal{L}_i, \tag{1.3}$$

where b_i is some positive constant associated with session i. Hayden offers a distributed algorithm for achieving Jaffe's desired rate vector.

The rationale behind (1.3) is more subtle than for (1.2). First, it allows us to establish different priorities among sessions, as characterized by the constant b_i. Second, it provides a buffer against transient flows which is sufficient to allow session i on link j to increase its rate by a factor of $(1+b_i)$, while still guaranteeing that $f_j \leq c_j$. Alternatively, it permits a new session to be added to the link, provided its rate is no greater than that of the most privileged session already using the link.

Gafni [11] further generalizes the feasible set considered by Jaffe. For each link $j \in \mathcal{L}$ and session $i \in \mathcal{S}$, he introduces functions $g_j : R^+ \rightarrow R^+$ and $b_i : R^+ \rightarrow R^+$. His objective is to generate a rate vector r such that the vector $(\ldots,b_i(r_i),\ldots)$ is a fair allocation over the set defined by

$$b_i(r_i)\leq g_j(c_j-f_j) \qquad \forall i\in\mathcal{S},\forall j\in\mathcal{L}_i \qquad (1.4.1)$$

and

$$r_i\geq 0 \qquad \forall i\in\mathcal{S} \qquad (1.4.2)$$

and

$$f_j\leq c_j \qquad \forall j\in\mathcal{L}. \qquad (1.4.3)$$

We will refer to the functions $g_j(\cdot)$ and $b_i(\cdot)$ as the link constraint functions and session constraint functions, respectively. For convenience, we will denote the vector $(\ldots,b_i(r_i),\ldots)$ by $\underline{b}(\underline{r})$.

In order to guarantee the existence of a unique fair allocation vector over this set, the following assumption is needed [12]:

Assumption 1.1: For all $j\in\mathcal{L}$, $g_j(\cdot)$ is monotonically non-decreasing, and for all $i\in\mathcal{S}$, $b_i(\cdot)$ is continuous, monotonically increasing and maps R^+ onto R^+.

We note that this assumption also implies the existence of $b^{-1}(\cdot)$. Unless otherwise noted, when discussing functions $g_j(\cdot)$ and $b_i(\cdot)$, we assume that Assuption 1.1 holds.

Gafni gives an algorithm that produces the desired rate vector, provided that an additional assumption about the link and session contraint functions is satisfied.

Assumption 1.2: For each $i \in \mathcal{S}$, $j \in \mathcal{L}_i$, the function $h_{ij}(\cdot)$, defined by $h_{ij}(f)=b_i^{-1}(g_j(f))$, is convex and differentiable on R^+, and satisfies $h_{ij}(0)=0$.

By allowing the function $b_i(\cdot)$ to be non-linear, we gain flexibility in making priority assignments. Gafni also provides some examples where it is desirable for $g_j(\cdot)$ to be non-linear as well. We summarize one such argument below.

Assume that the bit rate for each session i is a stochastic process with mean r_i and standard deviation $d_i r_i$, where $d_i \geq 0$. For each link j, define $D_j = \max_{i \in \mathcal{S}_j} d_i$. For a given link j, let k be the session in $\mathcal{S}_j$ with the maximum mean rate r_i. If we assume that $f_j \leq c_j$, then by the independence of the rates of different sessions, the standard deviation $\sigma(f_j)$ of the flow f_j satisfies

$$\begin{aligned}\sigma(f_j) &= \sqrt{\sum_{i \in \mathcal{S}_j} \sigma(r_i)^2} \\ &\leq D_j \sqrt{\sum_{i \in \mathcal{S}_j} r_i^2} \\ &\leq D_j \sqrt{(\sum_{i \in \mathcal{S}_j} r_i) r_k} \\ &\leq D_j \sqrt{c_j r_k} . \end{aligned} \tag{1.5}$$

If we choose

$$b_i(r)=r \tag{1.6.1}$$

and

$$g_j(f)=f^2/(c_j D_j^2), \tag{1.6.2}$$

then from (1.4) and (1.5), we have

$$\begin{aligned}\sigma(f_j) &\leq D_j\sqrt{c_j r_k}\\ &\leq D_j\sqrt{c_j g_j(c_j-f_j)}\\ &= c_j-f_j. \end{aligned} \qquad (1.7)$$

Hence, proper selection of the function $g_j(\cdot)$ can guarantee sufficient reserve capacity on each link to accomodate fluctuations in flow at least as great as the standard deviation of the flow.

We note that, while the class of feasible sets considered by Gafni is more general than the class of feasible sets considered by Hayden, the two classes are disjoint, because of Assumption 1.2.

In addition to seeking fair allocations over different feasible sets, Hayden and Gafni also use different methods of controlling the session input rates. Both algorithms are designed for synchronous operation, such that at each unit interval of time n, each session calculates a new input rate $r_i(n)$. For both algorithms, the rate vectors r(n) can be shown to converge as n goes to infinity, and the limit vector r is a fair allocation over the specified set.

In Hayden's algorithm, each link j calculates a control value $p_j(n+1)$ at time n+1, according to the equation

$$p_j(n+1)=p_j(n)+(a_jc_j-f_j(n))/W_j, \qquad (1.8.1)$$

where W_j is the number of sessions on (or "weight" of) link j, and a_j is some constant satisfying $0<a_j\leq 1$. Each session then adjusts its rate so that

$$r_i(n)= \min_{j\in\mathcal{L}_i} p_j(n). \qquad (1.8.2)$$

Hayden's algorithm can also be modified to give Jaffe's desired rate vector by changing the control update equation to

$$p_j(n+1)=p_j(n)+(a_jc_j-f_j(n)-p_j(n))/(W_j+1). \qquad (1.9)$$

While it has been shown that the rates $r_i(n)$ for Hayden's algorithm converge to the desired fair allocation, the control values for the links do not necessarily converge. If there exists a link such that all its sessions are controlled by other links, then the flow on that link will converge to some constant less than the capacity. In the attempt to bring the flow up to capacity, the link will increase its control at each update by $(c_j-f_j(n))/W_j$, and the control will grow to infinity. This can create serious problems when a new session joins the network.

Even though it has been shown that Hayden's algorithm converges to the desired rate vector, computer simulations of his algorithm exhibit distressing oscillations in the link flows when inputs are changing. We suspect this is caused by the failure of Hayden's model to accurately reflect delays in the network: the delay between the time that a link updates its control and its sessions learn the new value, and the delay between the time that a session changes its rate and the flow on any of its links reflects that change.

In Gafni's algorithm, at time n, each link j calculates for each of its sessions i, a control value $p_{ij}(n)$ given by

$$p_{ij}(n)=a_j(n)(h_{ij}(c_j-f_j(n))-r_i(n)) \qquad (1.10.1)$$

where

$$a_j(n)= 1/(1+\sum_{k\in S_j} h_{kj}'(c_j-f_j(n)) \qquad (1.10.2)$$

and $h_{ij}(\cdot)$ is defined as in Assumption 1.2, and $h'_{ij}(\cdot)$ is the derivative of $h_{ij}(\cdot)$. Each session then finds its new rate according to

$$r_i(n+1)=r_i(n)+ \min_{j\in \mathcal{L}_i} p_{ij}(n). \qquad (1.10.3)$$

If we define $b_i(r)=r$ and $g_j(f)=f$, then the rate vector as given by (1.4) is the same as Jaffe's, and (1.10.3) becomes

$$r_i(n+1)=r_i(n)+ \min_{j\in \mathcal{L}_i} (c_j-f_j(n)-r_i(n))/(W_j+1). \qquad (1.11)$$

Note the similarity of (1.11) and (1.9).

The essential difference between these two techniques is that, in Hayden's algorithm, memory of the past state resides with the links, while in Gafni's, past state information is stored by the sessions. In Hayden's algorithm the _links_ calculate their new control values in terms of the the past control values and the past flows (rates), while the sessions find their rates in terms of the present controls. In Gafni's algorithm, the _sessions_ calculate their new rates in terms of the past rates and the past controls, while the links find their controls in terms of the present flows (rates).

It is because new rates are calculated in terms of old, that Gafni's algorithm has the important property that the flows on the links are always less than or equal to the link capacities, provided that the flows were less than or equal to the capacities initially. Hayden's algorithm can only guarantee that flows are less than or equal to capacities in the steady state.

While this appears to be a serious flaw in Hayden's algorithm, it may provide certain advantages. Assume that the network in question is an integrated voice and data network, and that the stated capacity of a link c_j is not the link's true capacity but the portion of its capacity allocated for carrying voice packets. If the voice and data packets are queued separately, with voice being given priority, the algorithm could produce a rate assignment which would generate flows f_j in excess of c_j, without actually causing any voice packets to be queued. At the next iteration of the control update, the control for such a link would be greatly reduced and the voice flow f_j would drop below c_j, providing the extra capacity needed to transmit the data packets that were queued at the last step. Hence, the time average rate assignment for a given session is likely to be higher for Hayden's algorithm than for Gafni's.

Furthermore, Gafni's algorithm has not been shown to converge if the initial rates are chosen outside the feasible set. Hence, if random fluctuations in the session rates, or the initiation of a new session, cause flows to exceed capacity, there is no guarantee that the rates will return to values inside the feasible set. In the example above, where $g_j(\cdot)$ is given by (1.6.2), when the flows exceed capacity, the link constraint function does not even satisfy Assumption 1.1. Because Hayden's algorithm has been shown to converge from any initial control vector, it must eventually recover from such disturbances, if they are sufficiently infrequent.

Gafni's algorithm also has the disadvantage that each link must know the function $b_i(\cdot)$ for all of its sessions. This is not a serious drawback, though, since in practice there will probably be only a small number of different priority classes in use. The function $b_i(\cdot)$ will be the same for all members of a given priority class, so that the link only needs to calculate $p_{ij}(n)$ for each priority class.

As noted previously, both Hayden's and Gafni's algorithms are designed for synchronous operation, with all sessions updating their rates simultaneously, making actual implementation impractical. However, Gafni and Bertsekas [12] were able to show that Gafni's algorithm will produce a sequence of rate vectors that converge to the desired fair rate allocation, even under certain asynchronous conditions. Specifically, they consider an algorithm where a single session rate r_i is updated according to (1.10), and the flows are then updated to reflect the change in r_i. This process is repeated indefinitely, with each session updating in a fixed cyclic order.

1.4 Overview

In Chapter 1 of this thesis, we introduce the problem of flow control for packetized voice and introduce the idea of a fair rate allocation over a given feasible set. We describe previous work by Hayden and Gafni, each of whom developed distributed flow control algorithms for achieving fair rate assignments. We identify some problems associated with their algorithms.

In Chapter 2, we describe a method of categorizing flow control algorithms like Gafni's and Hayden's. We show how their two approaches can be merged to unite some of the advantages of each. In particular, we propose two algorithms which produce a fair rate vector for sets in Gafni's class of feasible sets (1.4), but without the need for the links to calculate separate controls for each priority class. We analyze one of these algorithms in detail.

In Chapter 3, we consider how the model of Chapter 1 fails to account for network delays, and describe some resulting difficulties. We then give an extended model which not only considers delay, but also asynchronous operation of the flow control algorithms. We introduce the idea of an update protocol, which permits a link to update its control only when the protocol is satisfied. We use update protocols to construct some asynchronous flow control algorithms: one that gives a fair rate vector over Hayden's feasible set and two that give fair rate vectors over sets in Gafni's class of feasible sets. For two of these algorithms, we prove that the generated control sequences converge to produce the appropriate fair rate vectors, given the assumptions of the asynchronous model.

In Chapter 4, we build on the work of Bertsekas, et. al., [13], [14], who have developed results that apply to general asynchronous algorithms. Bertsekas considers a system in which N processors find an element of a given solution set by iteratively computing estimates of the solution. Each processor receives feedback measurements from the system, and uses these measurements to update its current estimate. In Bertsekas' model, a processor may update its estimate at any time, asynchronously with respect to the other processors. We extend the model to include algorithms where updates times are restricted by update protocols. We give a theorem similar to Bertsekas', describing a class of such algorithms for which the estimate sequences converge. We use this result to give an alternate proof of the correct operation of one of the flow control algorithms given in Chapter 3. We also give a theorem that shows how a synchronous algorithm, taken from a given class of algorithms, can be implemented asynchronously by the addition of an appropriate update protocol.

In Chapter 5, we describe a computer program written to simulate the flow control algorithms of Chapter 3. The program simulates a network carrying voice traffic only, where each source-destination pair represents a voice conversation. At any given time, one member of each such pair is talking, and the other silent. We study the steady state behavior of the network by setting the average talk-spurt duration to infinity. For the static network, the algorithms behave much as predicted. We also set the average talk-spurt duration to a value representative of actual speech, to study how the network behaves under real-life conditions. The results of these simulations are inconclusive, and indicate that our model is not detailed enough to let us accurately predict the behavior of our algorithms in dynamic operation.

In Chapter 6, we summarize our results and give suggestions for further research.

Chapter 2

Generalized Synchronous Flow Control Algorithms

2.1 Options for Algorithm Design

As mentioned in section 1.3, the essential difference between Hayden's algorithm, given by (1.8) and Gafni's in (1.10), is that in Hayden's, state is stored by the links and in Gafni's, state is stored by the sessions. In addition to choosing where state memory resides, the algorithm designer must also consider how to allocate the burden of calculation. If the feasible set over which a fair allocation is sought is of the form given in (1.4), both the constraint functions $g_j(\cdot)$ and $b_i(\cdot)$ must appear somewhere in the update equations for the link controls or the session rates. Responsibility for calculations involving $g_j(\cdot)$ may be given to either the sessions or the links, and the same applies to $b_i(\cdot)$.

As an example of what this means, consider the following algorithm. Let

$$p_j(n+1)=p_j(n)+a_j(n)(c_j-f_j(n)-p_j(n)) \quad \forall j\in\mathcal{L} \qquad (2.1.1)$$

and

$$r_i(n)=\min_{j\in\mathcal{L}_i} b_i^{-1}(g_j(p_j(n))) \quad \forall i\in\mathcal{S} \qquad (2.1.2)$$

where $\{a_j(n)\}$ is some "appropriately" chosen sequence, designed to ensure the algorithm's convergence. This algorithm is the reverse of Gafni's: where his algorithm assigns memory to the sessions and calculation to the links, this algorithm assigns memory to the links and calculation to the sessions.

Of the eight possible ways to assign responsibilty for memory and calculation, two are clearly undesirable. An algorithm where calculations involving the link constraint functions are performed by the sessions, with the links doing the calculations involving the session constraint functions, obviously entails excessive overhead.

Ideally, we would like for a given link j to need to know only its own constraint function $g_j(\cdot)$, and for a given session i to need to know only its own constraint function $b_i(\cdot)$. Such an algorithm, with state memory assigned to the links, is given by

$$p_j(n+1)=p_j(n)+a_j(n)(g_j(c_j-f_j(n))-p_j(n)) \quad \forall j\in\mathcal{L} \qquad (2.2.1)$$

and

$$r_i(n)= \min_{j\in\mathcal{L}_i} b_i^{-1}(p_j(n)) \quad \forall i\in\mathcal{S} \qquad (2.2.2)$$

where the sequence $\{a_j(n)\}$ is chosen according to criteria discussed in section 2.4. We call this the generalized link memory algorithm.

Another such algorithm, with state memory assigned to the sessions, is given by

$$p_j(n)=g_j(c_j-f_j(n)) \quad \forall j \in \mathcal{L} \tag{2.3.1}$$

and

$$r_i(n+1)=r_i(n)+\min_{j \in \mathcal{L}_i} a_i(n)(b_i^{-1}(p_j(n))-r_i(n)) \quad \forall i \in \mathcal{S} \tag{2.3.2}$$

where $\{a_i(n)\}$ is some appropriately chosen sequence.

For both these algorithms, it is easy to see that if the rates and controls converge, they converge to values that satisfy (1.4.1). The problem remains to choose the sequences $\{a_i(n)\}$ or $\{a_j(n)\}$ in a manner that guarantees that the rates converge to a unique point, and further to show that that unique point is <u>fair</u> over the set defined by (1.4)

In general, it is not always possible to select a sequence that guarantees rate convergence. The choice of such a sequence depends on the link and the session constraint functions, and on which algorithm is being used.

In section 2.4, we give a definition for $a_j(n)$ and conditions on $g_j(\cdot)$ and $b_i(\cdot)$, such that the rates produced by the generalized link memory algorithm (2.2) can be shown to converge. We have not yet investigated this problem for the algorithm proposed in (2.3).

Before showing how to select the sequences $\{a_j(n)\}$ for the generalized link memory algorithm, we pause to discuss the nature of the limit points of the rate and control sequences, and to prove some theorems about the limit points.

2.2 The Fixed Points of the Rate and Control Update Equations

Before trying to prove that the control and rate sequences given by generalized link memory algorithm converge, we consider whether or not there exist control and rate vectors that are "fixed points" of (2.2), that is, we want $\underline{p}^*$ and $\underline{r}^*$ such that if $\underline{p}(n)=\underline{p}^*$ and $\underline{r}(n)=\underline{r}^*$, then $\underline{p}(n+1)=\underline{p}^*$ and $\underline{r}(n+1)=\underline{r}^*$. In this section, we give a centralized algorithm for finding the fixed points $\underline{p}^*$ and $\underline{r}^*$ of (2.2), and show that they are unique.

We will also see that the fixed rate vector $\underline{r}^*$ is the same for the algorithms given by (1.10), (2.1), (2.2) and (2.3). Since each of these algorithms was proposed to find the fair rate allocations over the same set, it is not surprising that they have the same $\underline{r}^*$.

In the next section, we show that $\underline{p}^*$ and $\underline{b}(\underline{r}^*)$ are fair allocations over the appropriate sets for the generalized link memory algorithm.

If there exist vectors $\underline{p}^*$ and $\underline{r}^*$ that are fixed points of (2.2), then

$$p_j{*}=g_j(c_j-f_j{*}) \qquad \forall j\in\mathcal{L} \tag{2.4.1}$$

and

$$r_i{*}=\min_{j\in\mathcal{L}_i} b_i^{-1}(p_j{*}) \qquad \forall i\in\mathcal{S} \tag{2.4.2}$$

where

$$f_j{*}=\sum_{i\in\mathcal{S}_j} r_i{*}. \tag{2.4.3}$$

Combining these three equations, we get

$$r_i{*}=\min_{j\in\mathcal{L}_i} b_i^{-1}(g_j(c_j-\sum_{k\in\mathcal{S}_j} r_k{*})) \tag{2.5}$$

It is easy to see that similar manipulations of (1.10), (2.1) and (2.3), all yield equation (2.5).

We now show how to uniquely construct **p***. For each link j define X_j such that

$$X_j=g_j(c_j-\sum_{i\in\mathcal{S}_j} b_i^{-1}(X_j)). \tag{2.6}$$

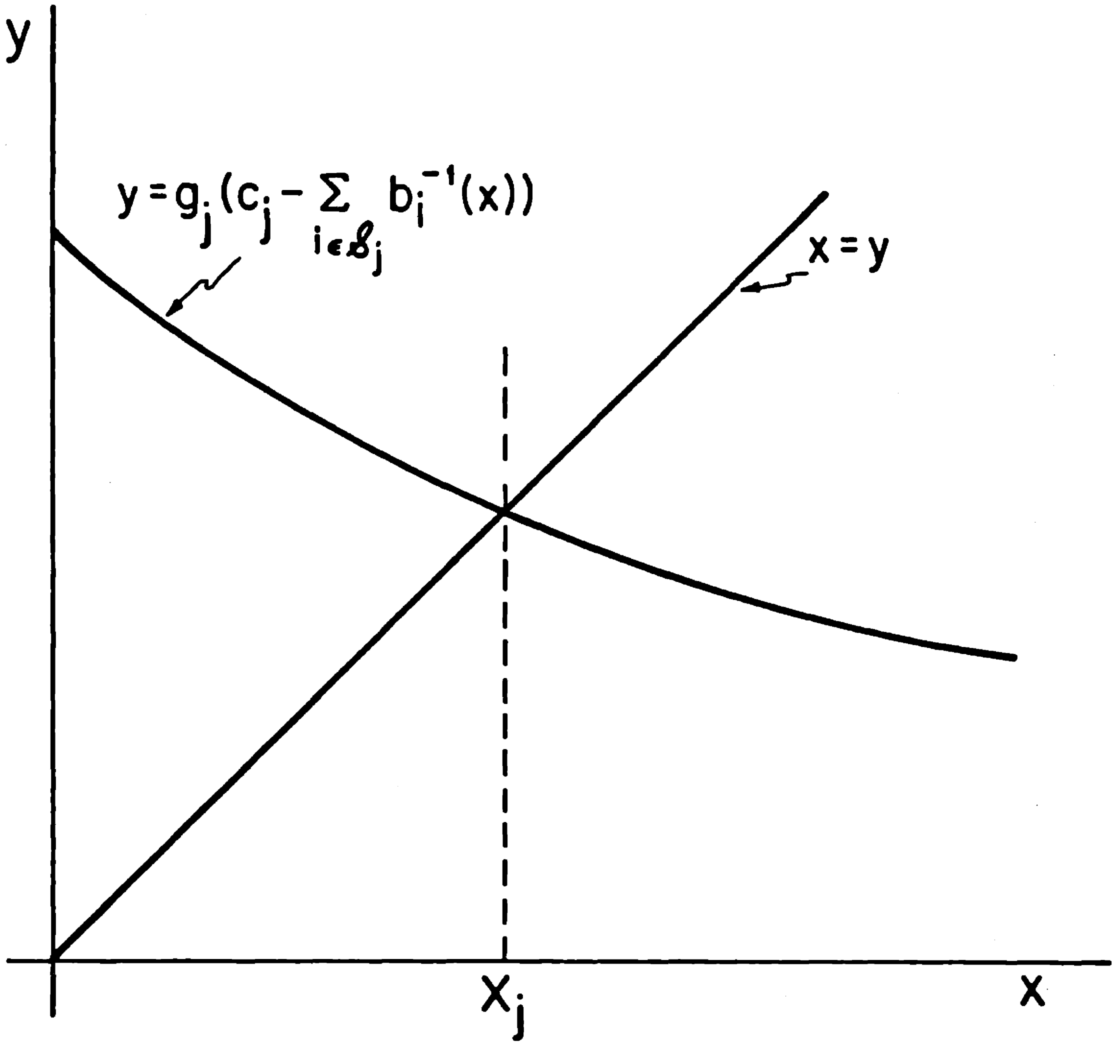
Assumption 1.1 guarantees that (2.6) has a unique solution. See Figure 2.1. Let $p^1=\min_{j\in\mathcal{L}} X_j$, let $\mathcal{L}^1$ be the set of links j for which $X_j=p^1$, and let $\mathcal{S}^1$ be the set of all sessions on links in $\mathcal{L}^1$.

Suppose p* exists. By (2.4) we have

$$p_j{*}=g_j(c_j-\sum_{i\in\mathcal{S}_j} b_i^{-1}(\min_{k\in\mathcal{L}_i} p_k{*})). \tag{2.7}$$

Therefore, by Assumption 1.1,

$$p_j{*}\geq g_j(c_j-\sum_{i\in\mathcal{S}_j} b_i^{-1}(p_j{*})). \tag{2.8}$$



The Fixed Point of $g_j(c_j -\quad b_i^{-1}(x))$

Figure 2.1

and so, from Figure 2.1, we see that $p_j^* \geq X_j$, for all $j \in \mathcal{L}$.

Bounding p_k^* by X_k in (2.7), we have,

$$p_j^* \leq g_j(c_j - \sum_{i \in \mathcal{S}_j} b_i^{-1}(\min_{k \in \mathcal{L}_i} X_k))$$

for $j \in \mathcal{L}^1$, and so

$$p_j^* \leq g_j(c_j - \sum_{i \in \mathcal{S}_j} b_i^{-1}(X_j))$$
$$= X_j. \qquad (2.9)$$

Thus if p^* exists, $p_j^* = \min_k X_k = p^1$ for all $j \in \mathcal{L}^1$.

Now we may rewrite (2.7) as

$$p_j^* = g_j(c_j - \sum_{i \in \mathcal{S}_j \cap \mathcal{S}^1} b_i^{-1}(p^1) - \sum_{i \in \mathcal{S}_j \setminus \mathcal{S}^1} b_i^{-1}(\min_{k \in \mathcal{L}_i} p_k^*)). \qquad (2.10)$$

Equation (2.10) suggests the following procedure. To find the next smallest p_j^*, construct the reduced network $\mathcal{N}' = (\mathcal{S}', \mathcal{L}')$, where $\mathcal{L}' = \mathcal{L} \setminus \mathcal{L}^1$, $\mathcal{S}' = \mathcal{S} \setminus \mathcal{S}^1$, and the capacities of the links are defined by

$$c_j' = c_j - \sum_{i \in \mathcal{S}_j \cap \mathcal{S}^1} b_i^{-1}(p^1). \qquad (2.11)$$

Now find X_j' for each link $\mathcal{L}'$, and $p^{1\prime} = \min_{j \in \mathcal{L}'} X_j'$. Repeat this procedure until all the coordinates of $\underline{p}^*$ have been found. Thus, by construction, p^* exists and is unique.

2.3 The Rate and Control Fixed Points are Fair Allocations

In this section, we show that $\underline{p}^*$ and $\underline{r}^*$, as found in section 2.2, are fair allocations over the sets specified below. This and the results of the last section imply the existence and uniqueness of a fair allocation rate over the set given by (1.4). This last result was also shown in [12].

Theorem 2.1 If $\underline{p}^*$ and $\underline{r}^*$ are the unique fixed points of (2.2), then $\underline{p}^*$ is a fair allocation over the set defined by

$$p_j \le g_j(c_j - \sum_{i\in\mathcal{S}_j} \min_{k\in\mathcal{L}_i} b_i^{-1}(p_k)) \quad \forall j \in \mathcal{L} \tag{2.12.1}$$

and $\underline{b}(\underline{r}^*)$ is fair over the set defined by

$$b_i(r_i) \le g_j(c_j - f_j) \quad \forall i \in \mathcal{S},\ \forall j \in \mathcal{L}_i. \tag{2.12.2}$$

Proof. As described in chapter 1, the fair allocation vector over a set X solves a nested hierarchy of problems. The first problem is to maximize the minimum coordinate of vectors in X. Next, we maximize the second minimum coordinate over all vectors which solve the first problem, and so on. Our algorithm for finding $\underline{p}^*$ and $\underline{r}^*$ solves for these vectors by just such a nested procedure, finding the minimum coordinate of each vector, then finding the next smallest coordinates, and so on. Hence, it is sufficient to show that the first iteration of the algorithm maximizes the minimum coordinates of $\underline{p}^*$ and $\underline{b}(\underline{r}^*)$. The "correctness" of the subsequent iterations follows by induction.

We claim that p^1, the minimum coordinate of $\underline{p}^*$ is the maximum minimal coordinate of any vector $\underline{p}$ in the set given by (2.12.1). Suppose otherwise. Then there must exist $\underline{q}$ in the feasible set with minimum coordinate $q^1 > p^1$. If that were so, we would have, for each link j,

$$\begin{aligned} p^1 &< q^1 \\ &\le q_j \\ &\le g_j(c_j - \sum_{i\in\mathcal{S}_j} b_i^{-1}(\min_{k\in\mathcal{L}_i} q_k)) \\ &\le g_j(c_j - \sum_{i\in\mathcal{S}_j} b_i^{-1}(q^1)) \\ &\le g_j(c_j - \sum_{i\in\mathcal{S}_j} b_i^{-1}(p^1)). \end{aligned} \qquad (2.13)$$

But this is a contradiction, since, for each $j\in\mathcal{L}^1$,

$$p^1 = g_j(c_j - \sum_{i\in\mathcal{S}_j} b_i^{-1}(p^1)). \qquad (2.14)$$

Now, since $b_i^{-1}(\cdot)$ is strictly increasing, (2.2.2) implies $b_i(r_i^*) = \min_{j\in\mathcal{L}_i} p_j^*$. Because p^1 is the minimum coordinate of $\underline{p}^*$, the minimum coordinate(s) of $\underline{b}(\underline{r}^*)$ must be $b_i(r_i^*)=p^1$, for each $i\in\mathcal{S}^1$. We claim that this choice maximizes the minimum coordinate of $\underline{b}(\underline{r}^*)$, since, if it did not, there must exist $\underline{b}(\underline{s})$ in the set given by (2.11.2) with minimum coordinate $b_m(s_m)>p^1$. But if that were so, we would have for each session i, for each link $j\in\mathcal{L}_i$,

$$\begin{aligned} p^1 &< b_m(s_m) \\ &\le b_i(s_i) \\ &\le g_j(c_j - \sum_{k\in\mathcal{S}_j} s_k) \\ &\le g_j(c_j - \sum_{k\in\mathcal{S}_j} b_k^{-1}(p^1)), \end{aligned} \qquad (2.15)$$

which is a contradiction. This completes the proof of Theorem 2.1.

Notice that we have not yet shown that $\underline{b}(\underline{r}^*)$ is fair over Gafni's feasible set (1.4), as desired. We cannot show this without making further restrictions on $b_i(\cdot)$ and $g_j(\cdot)$, since r_i^* might be negative for some i, and $\underline{b}(\underline{r}^*)$ might not even be an element of the set given by (1.4). This cannot happen when Assumption 1.2 holds. We give the following corollary.

Corollary 2.1. For each $i\in\mathcal{S}$, $j\in\mathcal{L}_i$, define $h_{ij}(x)=b_i^{-1}(g_j(x))$. If $h_{ij}(0)=0$, then $\underline{p}^*$ is fair over the set defined by

$$p_j \leq g_j(c_j - \sum_{i\in\mathcal{S}_j} \min_{k\in\mathcal{L}_i} b_i^{-1}(p_k)) \qquad \forall j\in\mathcal{L} \tag{2.16.1}$$

$$b_i^{-1}(p_j) \geq 0 \qquad \forall i\in\mathcal{S},\ \forall j\in\mathcal{L}_i \tag{2.16.2}$$

and

$$\sum_{i\in\mathcal{S}_j} \min_{k\in\mathcal{L}_i} b_i^{-1}(p_k) \leq c_j \qquad \forall i\in\mathcal{S},\ \forall j\in\mathcal{L}_i, \tag{2.16.3}$$

and $\underline{b}(\underline{r}^*)$ is fair over the set defined by (1.4).

In order to prove the corollary, we invoke the following lemma. The lemma is stated without proof, since it follows trivially from the definition of fair allocation.

Lemma 2.1 If a vector x is fair over a set X, and Y is a subset of X, then x is fair over Y, if and only if $x\in Y$.

Proof of Corollary. By the lemma, we need only show that $\underline{p}^*$ and $\underline{r}^*$ are elements of the appropriate sets.

Recalling (2.5), $\underline{r}^*$ is the solution to

$$r_i^* = \min_{j \in \mathcal{L}} h_{ij}(c_j - f_j^*), \tag{2.17}$$

and hence

$$\begin{aligned} f_j^* &= \sum_{i \in \mathcal{S}_j} \min_{k \in \mathcal{L}_i} h_{ik}(c_k - f_k^*) \\ &\leq \sum_{i \in \mathcal{S}_j} h_{ij}(c_j - f_j^*). \end{aligned} \tag{2.18}$$

Now suppose $f_j^* > c_j \geq 0$. Then

$$\begin{aligned} f_j^* &< \sum_{i \in \mathcal{S}_j} h_{ij}(0) \\ &= 0, \end{aligned} \tag{2.19}$$

which is a contradiction. Hence $f_j^* \leq c_j$, for all j. Since $f_j^* \leq c_j$, (2.17) gives us $r_i^* \geq 0$. That $\underline{p}^*$ satisfies (2.16) follows trivially from (2.4) and the fact that $\underline{r}^*$ satisfies (1.4). This completes our proof.

Note that the condition $h_{ij}(0)=0$ is sufficient to show that $\underline{r}^*$ is in the set defined by (1.4), but it is not necessary.

These theorems allows us to concentrate on finding a fair control vector, rather than a fair rate vector.

2.4 The Flow Control Algorithm

The genralized link memory algorithm is not completely specified until we describe how the sequences $\{a_j(n)\}$ are chosen. One obvious way to choose the sequences is to let

$a_j(n)=A_j(p_j(n))$, where $A_j(\cdot)$ is some real valued function.

In order to establish some conditions that $A_j(\cdot)$ should satisfy to guarantee the convergence of the controls, we consider a network consisting of a single link j. For such a network, (2.2) becomes

$$p(n+1)=p(n)+A(p(n))(g(c-\sum_{i\in\mathcal{S}} b_i^{-1}(p(n)))-p(n)). \qquad (2.20)$$

For convenience, define

$$G(x)=g(c-\sum_{i\in\mathcal{S}} b_i^{-1}(x)) \qquad (2.21)$$

and

$$H(x)=x+A(x)(G(x)-x). \qquad (2.22)$$

Then (2.20) becomes

$$p(n+1)=H(p(n)). \qquad (2.23)$$

It is well know that a sequence defined as in (2.23) will converge for any intial $\underline{p}(0)$, if

$$|H'(x)|<1 \quad \forall x \qquad (2.24)$$

and there exists x^* such that $H(x^*)=x^*$.

While the condition in (2.24) guarantees that the controls converge for a single link network, we are not able to show that (2.24) guarantees convergence for a more general network. We can, however, prove convergence for a multi-link network for which similar, but more restrictive, conditions hold.

<u>Theorem</u> <u>2.2</u> Let $\mathcal{N}=(\mathcal{S}\ \mathcal{L})$ be any network. Let $g_j(\cdot)$ and

$b_i(\cdot)$ satisfy Assumption 1.1. For each $j\in\mathcal{L}$, $S\subseteq\mathcal{S}_j$, define

$$G_j(x,c,S)=g_j(c-\sum_{i\in\mathcal{S}_j} b_i^{-1}(x)) \tag{2.25}$$

and

$$H_j(x,c,S)=x+A_j(x)(G_j(x,c,S)-x), \tag{2.26}$$

where $A_j(x)$is a continuous function such that $0<A_j(x)\leq 1$ for all x. Suppose, for each j, $g_j(\cdot)$ is uniformly continuous, and

$$0\leq \frac{\partial}{\partial x}H_j(x,c,S)<1, \tag{2.27}$$

for all x, for any $S\subseteq\mathcal{S}_j$ and c, $0\leq c\leq c_j$. Then, for any initial control vector $\underline{p}(0)$, the controls $\underline{p}(n)$ and rates $\underline{r}(n)$ given by (2.2), with $a_j(n)=A_j(p_j(n))$, converge to fair allocations over the sets given by (2.11).

The proof of Theorem 2.2 is deferred to Chapter 3, where we will see that it is a special case of a more general theorem relating to asynchronous flow control algorithms.

The conditions of Theorem 2.2 are somewhat restrictive and we believe that the controls and rates will converge for algorithms where $A_j(\cdot)$, $g_j(\cdot)$ and $r_i(\cdot)$ are less constrained. Specifically, we conjecture that condition (2.27) can be replaced by

$$\left|\frac{\partial}{\partial x}H_j(x,c,S)\right|<1, \tag{2.28}$$

but we have been unable to prove so.

Let us consider the restrictions that (2.27) places on $A_j(\cdot)$, $g_j(\cdot)$ and $b_i(\cdot)$. While it is difficult to describe the entire class of functions $H_j(\cdot,\cdot)$ such that $A_j(\cdot)$ may be chosen to satisfy (2.27), at least one sub-class is easily identifiable.

Consider the class of functions $H_j(\cdot,\cdot)$ such that $A_j(\cdot)$ may be chosen to be a constant, that is, $A_j(x)=A_j$, and such that (2.27) is satisfied. We assume that $0<A_j<1$. Then (2.27) becomes

$$-1\leq A_j(\frac{\partial}{\partial x}G_j(x,c,S)-1)<0. \qquad (2.29)$$

By Assumption 1.1, $\frac{\partial}{\partial x}G_j(x,c,S)$ is negative for all x, c and S, and hence the right inequality of (2.29) is always satisfied. Rearranging the left inequality, we get

$$1-1/A_j\leq \frac{\partial}{\partial x}G_j(x,c,S). \qquad (2.30)$$

Hence, for those functions $G_j(x,c,S)$ whose partial derivatives with respect to x are bounded below, the conditions of (2.27) can always be met for small enough A_j.

While (2.30) may seem a bit restrictive, we make the following observation. Suppose that we are given two sets of functions $g_j(\cdot)$ and $g_j'(\cdot)$ that satisfy Assumption 1.1, and such that $g_j(x)=g_j'(x)$ for $x\in[0,c_j]$. Then regardless of how the functions may differ outside that interval, the fair rate allocation over the set defined by (1.4) and $g_j(\cdot)$ is the same as the fair rate allocation defined by (1.4) and $g_j'(\cdot)$.

Hence, we may "tailor" the functions $g_j(\cdot)$ any way we like outside $[0,c_j]$, in order to satisfy (2.30), without affecting the point to which the algorithm converges. Even so, there may be some functions $b_i(\cdot)$ and $g_j(\cdot)$ for which (2.30) will not hold.

2.5 An Example

In the last section we described how the functions $g_j(\cdot)$ may be tailored outside the interval $[0,c_j]$ to give a function which will satisfy the constraints of Theorem 2.2. In this section, we give an example of this technique.

Suppose we are interested in finding a fair allocation over the set given by (1.4), using functions $b_i(\cdot)$ and $g_j(\cdot)$ as defined by (1.6). Recall that such a fair allocation has the property that the excess capacity on any link is sufficient to handle a fluctuation in flow as great as the standard deviation of the flow. For convenience define $k_j=1/(c_jD_j^2)$. Then $g_j(f)=k_jf^2$. This choice of $g_j(\cdot)$ is clearly unsatisfactory, since it is neither monotonically non-decreasing, nor uniformly continuous. We propose, instead, that $g_j(\cdot)$ be defined as

$$g_j(f)=\begin{cases}0 & \text{when } f<0\\ k_jf^2 & \text{when } 0\leq f\leq c_j\\ 2k_jc_jf-k_jc_j^2 & \text{when } f>c_j.\end{cases} \qquad (2.31)$$

Now,

$$H_j(x,c,S)=x+A_j(g_j(c-W_S x)-x), \qquad (2.32)$$

where W_S is the number of sessions in S. So,

$$\frac{\partial}{\partial x}H_j(x,c,S)=1-A_j(W_S G_j'(c-W_S x)+1). \qquad (2.33)$$

Since

$$g_j'(f)=\begin{cases} 0 & \text{when } f<0 \\ 2k_j f & \text{when } 0\leq f\leq c_j \\ 2k_j c_j & \text{when } f>c_j \end{cases} \qquad (2.34)$$

we have, for all x,

$$0\leq g_j'(x)\leq 2k_j c_j. \qquad (2.35)$$

Combining (2.33) and (2.35), we get

$$1-A_j(1+2k_j c_j W_S)\leq H_j(x,c,S)<1. \qquad (2.36)$$

Hence, we choose

$$A_j=1/(1+2k_j c_j W_S) \qquad (2.37)$$

and the conditions of Theorem 2.2 are satisfied.

Chapter 3

Asynchronous Flow Control Algorithms

As mentioned earlier, the system model given in Chapter 1 fails to accurately describe the operation of flow control algorithms by ignoring communication delays. In this chapter, we describe in detail how the model fails, and then give an improved model which allows for feedback delays and asynchronous operation. We introduce the idea of an update protocol and give some examples. We give an improved flow control algorithm, together with a proof that the rates and controls it produces converge to the fair allocation over Hayden's feasible set, under the assumptions of the asynchronous model. Finally, we modify the algorithm to produce fair rates and controls over the more general feasible set defined by (2.11).

3.1 Feedback Delays

In section 1.2 we made the assumption that data propagates instantaneously through the network, allowing us to define the flow on a link at time t as

$$f_j(t)= \sum_{i \in \mathcal{S}_j} r_i(t) \qquad \forall j \in \mathcal{L}. \tag{3.1}$$

We also assumed that at time t, each session i knows the current value of

$$\min_{j \in \mathcal{S}_i} p_j(t)$$

or

$$\min_{j \in \mathcal{S}_i} p_{ij}(t)$$

as required for the calculation of $r_i(t)$.

In practice, however, the concepts of instantaneous flow and rate are not well defined. The presence of queues distorts the input rates of the sessions as seen by the links, and even in the absence of queues, propagation delays prevent the links from knowing the sessions' current input rates. Hence, we cannot expect these relations to hold at all times t. However, for the flow control algorithms previously discussed, where rate and control updates take place at t_n, n=0,1,..., we require only that

$$f_j(t_n)= \sum_{i \in \mathcal{S}_j} r_i(t_n) \qquad \forall j \in \mathcal{L}, \tag{3.2}$$

and that each session i knows at time t_n, the current value of

$$\min_{j \in \mathcal{S}_i} p_j(t_n)$$

or

$$\min_{j \in \mathcal{S}_i} p_{ij}(t_n).$$

We outline below a method for guaranteeing that (3.2) holds. The other condition can be met only by careful synchronization of the links and sessions, when selecting the update times t_n.

In order to guarantee that (3.2) holds for all t_n we assume that the links calculate their flows by summing the rates of their sessions, where the rates are communicated to the links by the sessions. While this eliminates the problem of rate distortion, there will still be delays. Communicating the rates requires transmitting additional data between links and sessions, but guarantees that (3.2) holds, provided that the interval between updates is long enough to allow the necessary exchange of information.

An alternative to this would be for each link to observe the amount of traffic it carried over some recent interval of time, and use that as its flow. Observing the flow has the advantages of simplicity and low overhead, but makes it impossible to enforce the condition (3.2), because of rate distortion by the queues.

Hayden's [9] simulation of a network using his flow control algorithm assumed that flows were observed. We believe that this contributed to the oscillations in flow that his simulation displayed. Another contributing factor is the delay between the time that a link updates its control, and its sessions learn the new control value. To see how such oscillations might arise, we consider the following examples.

Suppose we have a network consisting of a single link, with capacity $c=1$, serving a single session. If we seek a fair allocation over the set given in (1.2) with $a=1$, the update equations for Hayden's algorithm are

$$p(n+1)=p(n)+c-f(n) \tag{3.3.1}$$

and

$$r(n)=p(n). \tag{3.3.2}$$

Suppose that the delays are such that $f(n)=p(n-1)$, instead of $f(n)=p(n)$. Then

$$p(n+1)=p(n)-p(n-1)+1. \tag{3.4}$$

For $p(0)=0$ and $p(1)=1$, the subsequent controls are given in Table 3.1. Obviously, the controls cycle forever, and do not converge.

For the same network, suppose we want a fair allocation over the set in (1.4), with $g(f)=f$ and $b(r)=r$. Using Gafni's algorithm, the update equations are

$$p(n)= .5(c-f(n)-r(n)) \tag{3.5.1}$$

and

$$r(n+1)=r(n)+p(n). \tag{3.5.2}$$

If the flow is communicated, $f(n)=r(n)$. However, feedback delay can cause the session to learn the link control value late, so that

$$\begin{aligned} r(n+1)&=r(n)+p(n-1) \\ &=r(n)+ .5(1-2r(n-1)) \\ &=r(n)-r(n-1)+ .5 \ . \end{aligned} \tag{3.6}$$

For $r(0)=0$ and $r(1)=.5$, the subsequent rates are given in Table 3.2. Again we see that the rates cycle, and do not

n	p(n)
0	0
1	1
2	2
3	2
4	1
5	0
6	0
7	1

Table 3.1

Divergent Example for Hayden's Algorithm

n	r(n)
0	0
1	.5
2	1
3	1
4	.5
5	0
6	0
7	.5

Table 3.2

Divergent Example for Gafni's Algorithm

converge.

These simple examples show how important it is to account for feedback delay in our system model.

3.2 The Asynchronous System Model

In this section, we give a model for studying systems with feedback delay which use asynchronous flow control algorithms. The model assumes that flows are communicated and that state memory is assigned to the links. The model can be easily changed to allow algorithms with link memory.

At a given time t, each link j has a control value $p_j(t)$ and a flow $f_j(t)$, and each session i has rate $r_i(t)$. We assume that the system begins operation at t=0, that is, $p_j(t)=0$ and $r_i(t)=0$ for $t<0$. At t=0, the links and sessions choose some arbitrary initial controls $p_j(0)$ and rates $r_i(0)$.

For each link j, session $i\in\mathcal{L}_j$ has an apparent rate $r_{ij}(t)$, which is the most recent rate communicated to link j by session i at time t. The apparent rate $r_{ij}(t)$ may differ from $r_i(t)$ because of communication delays. Hence we have

$$r_{ij}(t)=r_i(t-d_{ij}(t)), \tag{3.7}$$

where $d_{ij}(t)$ is the delay described above. We will call $d_{ij}(t)$ the propagation delay. With this definition of rate, flow is defined as

$$f_j(t) = \sum_{i \in \mathcal{S}_j} r_{ij}(t). \tag{3.8}$$

When the system has been running long enough for a session i to have received feedback from each of the links in its path, the rate $r_i(t)$ at which a session sends is chosen as the minimum of the control values of the links. Again, because of communication delays, the most recent values of the control for link j known by session i may not be current. Hence,

$$r_i(t) = \begin{cases} \min_{j \in \mathcal{L}_i} p_j(t - D_{ij}(t)) & \text{when } t \geq D_{ij}(t) \ \forall j \in \mathcal{L}_i \\ r_i(0) & \text{when } 0 \leq t < D_{ij}(t) \text{ for some } j \in \mathcal{L}_i \\ 0 & \text{when } t < 0. \end{cases} \tag{3.9}$$

We will call $D_{ij}(t)$ the feedback delay.

We make two assumptions about the processes $d_{ij}(t)$ and $D_{ij}(t)$. First, for $t_1 \leq t_2$

$$t_1 - d_{ij}(t_1) \leq t_2 - d_{ij}(t_2) \tag{3.10.1}$$

and

$$t_1 - D_{ij}(t_1) \leq t_2 - D_{ij}(t_2). \tag{3.10.2}$$

This guarantees that new information is not replaced by old.

Second, for any times t_{ij}^0 and T_{ij}^0, there must exist $t_{ij}^1 \geq t_{ij}^0$ and $T_{ij}^1 \geq T_{ij}^0$ such that

$$t_{ij}^0 \leq t_{ij}^1 - d_{ij}(t_{ij}^1) \tag{3.11.1}$$

and

$$T_{ij}^0 \leq T_{ij}^1 - D_{ij}(T_{ij}^1). \tag{3.11.2}$$

This assumption guarantees that the links and sessions will never stop receiving new information about the rates and controls current in the network. That is, if a link j has control $p_j(t)$ at some time t, then each of its sessions will eventually learn the value of the link's control at t or some later time. Similarly, the apparent rate of session i on link j must eventually reflect the changes in the session's input rate $r_i(t)$.

Together, these two assumptions guarantee that the system eventually changes from its initial conditions. If we let $T_{ij}^0=0$, then (3.11.2) guarantees the existence of a time $t_0>0$ such that $D_{ij}(t_0)\leq t_0$. Furthermore, by (3.10.2),

$$\begin{aligned} 0 &\leq t_0 - D_{ij}(t_0) \\ &\leq t - D_{ij}(t) \end{aligned} \tag{3.12}$$

for all $t\geq t_0$. Hence $D_{ij}(t)\leq t$ for $t\geq t_0$. Thus, for large enough t, (3.9) becomes

$$r_i(t) = \min_{j\in\mathcal{L}_i} p_j(t-D_{ij}(t)) \tag{3.13}$$

and (3.7) becomes

$$r_{ij}(t) = \min_{k\in\mathcal{L}_i} p_k(t-d_{ij}(t)-D_{ik}(t-d_{ij}(t))). \tag{3.14}$$

The controls for the links are updated as follows. For each link j, we are given a monotonic increasing sequence $\{t_j^n\}$ for $n\geq 0$, with $t_j^0\geq 0$. We define $p_j^n=p_j(t_j^n)$ and $f_j^n=f_j(t_j^n)$. We assume that $p_j(t)=p_j(0)=p_j^0$ for $0<t\leq t_j^0$.

Then t_j^n is the time of (n+1)st control update for link j, and

$$p_j^{n+1}=P_j(\underline{p}_j^n,\{r_{ij}(t_j^n):i\in\mathcal{L}_j\},c_j,W_j) \tag{3.15}$$

where P_j is the control update function for link j, $\underline{p}_j^n=(p_j^n,p_j^{n-1},\ldots,p_j^0)$, c_j and W_j are the capacity and weight of link j, respectively. The control for link j at time t, for $t_j^{n-1}<t\leq t_j^n$, is $p_j(t)=p_j^n$. See Figure 3.1.

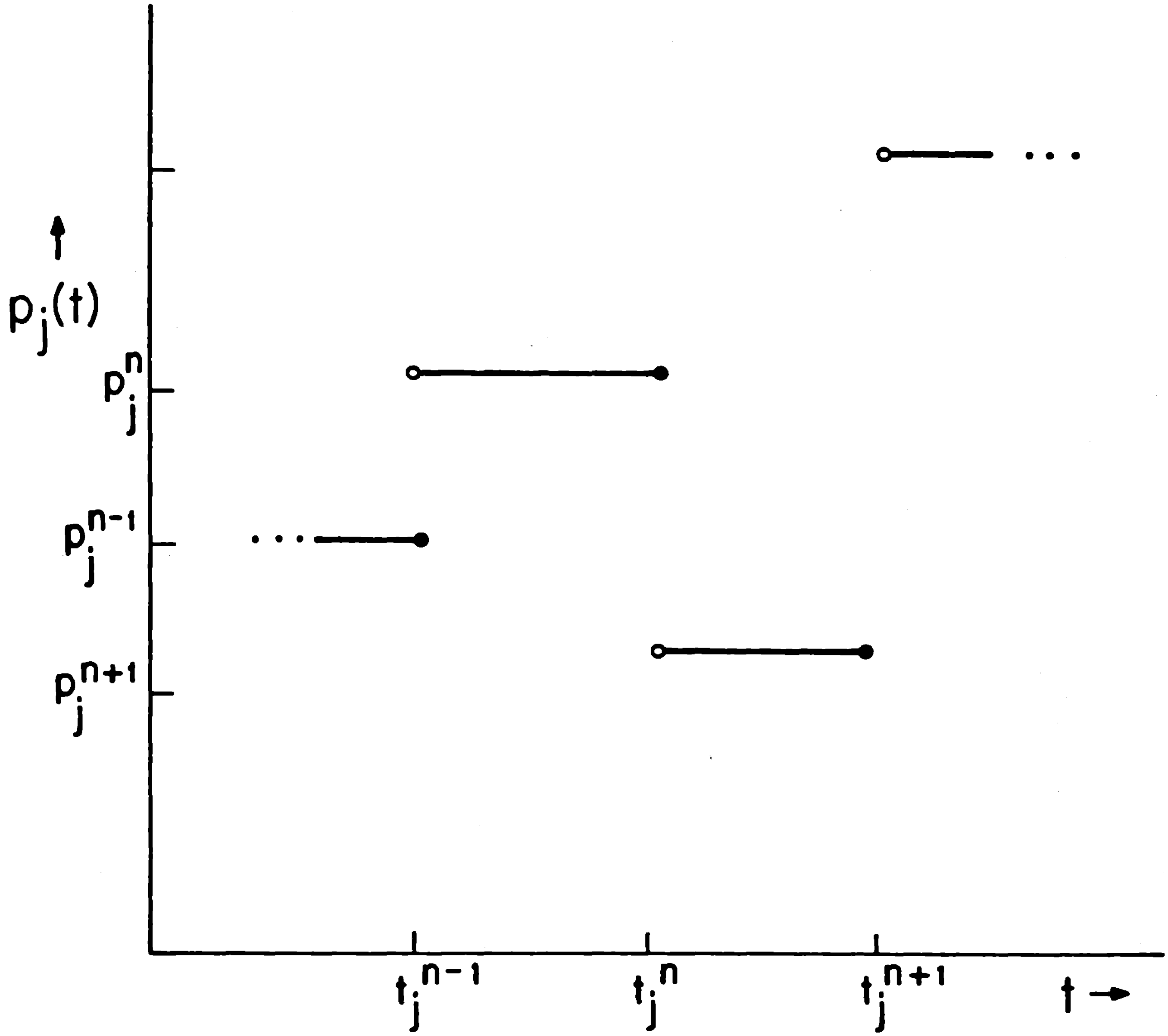
Note that Hayden's algorithm is a special case of this model, obtained by letting $d_{ij}(t)=D_{ij}(t)=0$ for all t, and $t_j^n=n$ for all n, for each j, and defining

$$\begin{aligned} P_j(\underline{p}_j^n,\{r_{ij}(t_j^n):i\in\mathcal{L}_j\},c_j,W_j)&=p_j^n+(c_j-\sum_{i\in\mathcal{L}_j}r_{ij}(t_j^n))/W_j \\ &=p_j^n+(c_j-f_j^n)/W_j. \end{aligned} \tag{3.16}$$

Since, in general, $d_{ij}(t)\neq 0$ and $D_{ij}(t)\neq 0$, and because it is difficult to synchronize links such that $t_j^n=n$, it is useful to ask for what values of $d_{ij}(t)$, $D_{ij}(t)$ and t_j^n we can expect the rates $r_i(t)$ to converge to some desired set of values.

3.3 Update Protocols for Asynchronous Flow Control Algorithms

In this section we consider asynchronous flow control algorithms such that the system can select the times t_j^n according to some established criteria. Such a set of criteria is called an update protocol. We examine some possible update protocols, in conjunction with different



Link Control as a Function of Time

Figure 3.1

update functions.

Let us consider an algorithm using Hayden's control update function, but with $d_{ij}(t)\neq 0$, $D_{ij}(t)\neq 0$ and $t_j^n\neq n$. If upper bounds are known for $d_{ij}(t)$ and $D_{ij}(t)$, it might be possible to eliminate the effects of these delays on the controls and rates by waiting "long enough" between control updates. That is, we would like to choose t_j^n such that

$$r_{ij}(t_j^n)=r_i(t_j^n)=\min_{k\in\mathcal{L}_i} p_k(t_j^n). \tag{3.17}$$

In this case, the rates and controls would be identical to those for a system in which $d_{ij}(t)=D_{ij}(t)=0$.

Since the proof of convergence for Hayden's algorithm relies very little on the synchronous properties of his algorithm, his proof is easily modified to show convergence for an asynchronous algorithm where (3.17) holds.

Unfortunately, the condition given in (3.18) is difficult to meet, since it requires that

$$\begin{aligned} r_{ij}(t_j^n)&=r_i(t_j^n-d_{ij}(t_j^n)) \\ &=\min_{k\in\mathcal{L}_i} p_k(t_j^n-d_{ij}(t_j^n)-D_{ik}(t_j^n-d_{ij}(t_j^n))) \\ &=\min_{k\in\mathcal{L}_i} p_k(t_j^n). \end{aligned} \tag{3.18}$$

Necessary conditions for (3.18) to hold are complicated, but it is clearly sufficient that

$$p_k(t_j^n)=p_k(t_j^n-d_{ij}(t_j^n)-D_{ik}(t_j^n-d_{ij}(t_j^n))) \tag{3.19}$$

for each $k\in\mathcal{L}_i$. If we choose t_j^n such that

$$t_j^n - t_j^{n-1} > d_{ij}(t_j^n) + D_{ij}(t_j^n - d_{ij}(t_j^n)) \qquad (3.20)$$

then (3.19) is satisfied for k=j. But to guarantee (3.19) for $k\neq j$, it is necessary that there be no updates to the control for link k between the time that session i learns the latest value of p_k^n and the time that the flow on link j reflects any possible change in $r_i(t)$ due to a change in p_k^n. This could be accomplished by making the links and sessions update during alternating intervals. That is, during a link update interval all links will update their controls, while sessions may not update their rates. The reverse is true for a session update interval. This scheme has the disadvantages of being slow and requiring additional communications to inform sessions and links of the ends of the update intervals.

Another approach would be to perform updates only when all the sessions on a link are either aware of the link's current control, or are being controlled by another link whose control is smaller. That is, choose t_j^n such that

$$r_{ij}(t_j^n) \leq p_j(t_j^n). \qquad (3.21)$$

If the links are able to observe the rates of their individual sessions, then the condition of (3.21) is easy to enforce, provided that a sequence of times $\{t_j^n\}$ satisfying (3.21) actually exists. Fortunately, the delay conditions (3.10) and (3.11) guarantee the existence of such a sequence.

We want to show that for each link j and any time t_1, there exists $t_2 \geq t_1$ such that $r_{ij}(t_2) \leq p_j(t_2)$, for each $i \in \mathcal{S}_j$. We will need the following theorem:

<u>Theorem 3.1</u>. If $d_{ij}(t)$ and $D_{ij}(t)$ satisfy (3.10) and (3.11), then for any t^0 there exists $t^1 \geq t^0$ such that for all $t \geq t^1$,

$$t-d_{ij}(t)-D_{ik}(t-d_{ij}(t)) \geq t^0. \tag{3.22}$$

for all $j \in \mathcal{L}$, $i \in \mathcal{S}_j$, $k \in \mathcal{L}_i$.

The proof of Theorem 3.1 is given in Appendix A.

Now suppose that no such t_2 existed. Then no link updates could take place after t_1. Now $p_j(t_1)=p_j{}^n$ for some n, and so $p_j(t)=p_j{}^n$ for all $t \geq t_1$. But by Theorem 3.1, there must exist $t' \geq t_1$ such that for all $t \geq t'$

$$t-d_{ij}(t)-D_{ij}(t-d_{ij}(t))) \geq t_1. \tag{3.23}$$

By (3.14), for large enough t,

$$\begin{aligned} r_{ij}(t) &= \min_{k \in \mathcal{L}_i} p_k(t-d_{ij}(t)-D_{ik}(t-d_{ij}(t))) \\ &\leq p_j(t-d_{ij}(t)-D_{ij}(t-d_{ij}(t))) \\ &= p_j(t_1) \\ &= p_j(t), \end{aligned} \tag{3.24}$$

which is a contradiction. Hence, an infinite sequence of times $\{t_j{}^n\}$ must exist such that (3.21) is satisfied.

Unfortunately, it is not possible to guarantee convergence of the rates for a network using the control update function in (3.16) and the update protocol in (3.21). A counter example is easily constructed using the following argument. Suppose that for some link j at time t_j^n, (3.21) is satisfied and $f_j^n < c_j$. Then $p_j^{n+1} = p_j^n + (c_j - f_j^n)/W_j > p_j^n$. Now at t_j^n+e, $p_j^{n+1} > p_j^n$, and if e is taken small enough, the rates will not have had sufficient time to change and $r_{ij}(t_j^n+e) = r_{ij}(t_j^n) < p_j^n < p_j^{n+1}$. Hence the link may update again "immediately". By updating the control arbitrarily often, the control can be increased to any desired value.

An obvious approach to fixing this problem is to require that

$$t_j^{n+1} - t_j^n \geq x \tag{3.25}$$

for some positive x, for all $j \in \mathcal{L}$, for all n. Another approach would be to put an upper bound on the value of p_j^n. While it seems likely that either of these conditions would guarantee convergence of the rates, we have not been able to prove this.

3.4 An Asynchronous Flow Control Algorithm

Fortunately, it is possible to prove convergence of the rates for a network using this update function:

$$p_j^{n+1} = \max_{i \in \mathcal{S}_j} r_{ij}(t_j^n) + (c_j - f_j^n)/W_j \tag{3.26}$$

along with the update protocol given in (3.21), but without any additional requirements.

The new update function has the further advantage over Hayden's that it guarantees convergence of the controls as well as the rates. For Hayden's algorithm the controls may diverge, since the algorithm finds the fair control allocation over the set defined by

$$\sum_{i\in \mathcal{S}_j} \min_{k\in \mathcal{L}_i} p_k \le c_j \qquad \forall j. \tag{3.27.1}$$

Thus, if a link controls none of its session, the fair control for that link is infinity. Our algorithm finds the fair allocation over the set defined by

$$\sum_{i\in \mathcal{S}_j} \min_{k\in \mathcal{L}_i} p_k + W_j(p_j - \max \min p_k) \le c_j \qquad \forall j. \tag{3.27.2}$$

It is easy to see that the fair rate vector produced by either (3.27.1) or (3.27.2) is the same, for if a link j controls any of its sessions, p_j=max min p_k, and the two conditions are equivalent. Otherwise, it does not matter what control link j is assigned, since it does not affect the rates.

We can find the fair control vector over the set defined by (3.27.2) by a global procedure almost identical to Hayden's. We begin by finding the bottleneck link j, assigning it $p_j*=c_j/W_j$ and assigning its sessions rates equal to p_j*. We then form a reduced network by deleting that link and its sessions from the original network, and reducing the capacity of the remaining links by the appropriate amount. We repeat this procedure, finding the new bottleneck link and reducing the network, until all links have controls assigned. However, we may eventually find that our reduced network contains a link that has no users. The fair allocation over (3.27.1) sets the control for such a link to infinity, whereas the fair allocation over (3.27.2) sets it to

$$p_j^* = \max_{i \in \mathcal{S}_j} \min_{k \in \mathcal{L}_i} p_k + (c_j - \sum_{i \in \mathcal{S}_j} r_i^*)/W_j. \qquad (3.27.3)$$

We may interpret (3.26) as follows. Hayden's algorithm fails to produce controls that converge for two reasons. In the absence of any update protocol a link may update its control before all of its sessions have learned the current control value, thereby producing oscillations in the control. Furthermore, when a link controls none of its sessions, it increases its control without bound. The update protocol overcomes the first problem by letting the link pretend that all its sessions know its most recent value. The link simply assumes that those sessions that are sending at a lower rate are being controlled by some other link. The update function overcomes the second problem by making the link pretend that its current control is actually the rate of its fastest session. Thus, whenever an update takes place, the link is effectively controlling at least one of its sessions.

Next we prove that the algorithm given by (3.21) and (3.26) produces input rates and link controls that converge to the desired values. We give the following theorem:

<u>Theorem</u> <u>3.2</u>. Let $\mathcal{N}=(\mathcal{S},\mathcal{L})$ be any network. Let $p_j(t)$, $f_j(t)$, $r_i(t)$, $r_{ij}(t)$, $p_j{}^n$ and $f_j{}^n$ be given according to the

asynchronous flow control model, where the control update function is given by (3.26), the control update times t_j^n satisfy (3.21), and the delay functions $d_{ij}(t)$ and $D_{ij}(t)$ satisfy (3.10) and (3.11). Then, for any initial control vector $\underline{p}^0$ and initial rate vector $\underline{r}^0$, $\lim_{t\to\infty} \underline{p}(t)=\underline{p}^*$ and $\lim_{t\to\infty} \underline{r}(t)=\underline{r}^*$, where $\underline{p}^*$ is the fair allocation over the set defined by (3.27.2) and $\underline{r}^*$ is fair over the set defined by (1.2).

In order to prove Theorem 3.2, we will need the following lemma. The lemma says that the control for a "bottle-neck" link $j\in\mathcal{L}^1$ converges to p_j^*, in a network where the flows are perturbed from their true values, provided that the perturbations eventually go to zero. This implies that the control for $j\in\mathcal{L}^1$ of a network with no perturbations must converge, and the rates of its sessions must also converge. To the other links, this network is indistinguishable from a reduced network obtained by deleting link j and all its sessions from the original network, decreasing the capacity of each remaining link k by $\sum_{i\in\mathcal{S}^1\cap\mathcal{S}^k} r_i^*$, and perturbing the flow $f_k(t)$ by $\sum_{i\in\mathcal{S}^1\cap\mathcal{S}^k}(r_i^*-r_i(t))$. Since the perturbations in the flows go to zero, by the lemma, the controls for the bottle-neck links of the reduced network must also converge. By induction, we may then show that the controls for all the links converge.

<u>Lemma 3.2</u>. Let $\mathcal{N}=(\mathcal{S},\mathcal{L})$ be any network. For each $j\in\mathcal{L}$, let

$\hat{W}_j$ be some constant satisfying $\hat{W}_j > W_j$, let $\{e_j^n\}$ be some sequence that converges to 0, and let $\{t_j^n\}$ be a sequence that satisfies the conditions given below. Let $\hat{\underline{p}}(t)=\underline{0}$ and $\hat{\underline{r}}(t)=\underline{0}$ for $t<0$ and choose any initial vectors $\hat{\underline{p}}(0)$ and $\hat{\underline{r}}(0)$. Let $\hat{p}_j(t)=\hat{p}_j(0)=\hat{p}_j^0$ for $0<t\leq t_j^0$ and define

$$\hat{p}_j^{n+1}= \max_{i\in\mathcal{S}_j} \hat{r}_{ij}(t_j^n)+(c_j-\hat{f}_j^n-e_j^n)/\hat{W}_j \tag{3.28}$$

where $\hat{f}_j^n$, $\hat{r}_{ij}(t)$, $\hat{r}_i(t)$ are defined as in (3.7) - (3.9), and $\hat{p}_j(t)=\hat{p}_j(t_j^n)$ for $t_j^{n-1}<t\leq t_j^n$. Assume that $d_{ij}(t)$ and $D_{ij}(t)$ satisfy (3.10) and (3.11). Further suppose that the sequence $\{t_j^n\}$ is chosen so that

$$\hat{r}_{ij}(t_j^n)\leq\hat{p}_j(t_j^n) \qquad \forall j\in\mathcal{L},\ \forall i\in\mathcal{S}_j,\quad n\geq 0, \tag{3.29.1}$$

and that the sequence $\{e_j^n\}$ is chosen so that

$$\hat{p}_j^n>0 \qquad\qquad \forall n\geq 1,\ \forall j\in\mathcal{L}. \tag{3.29.2}$$

Let $p^1= \min_{j\in\mathcal{L}} c_j/W_j$. Define $\mathcal{L}^1$ and $\mathcal{S}^1$ in the obvious manner. Then $\lim_{t\to\infty} \hat{p}_j(t)=p^1$ and $\lim_{t\to\infty} \hat{r}_{ij}(t)=p^1$ for all $j\in\mathcal{L}^1$, $i\in\mathcal{S}^1$. Furthermore,

$$\liminf_{t\to\infty} \hat{p}_j(t)\geq z_j^* \tag{3.29.3}$$

and

$$\liminf_{t\to\infty} \hat{r}_{ij}(t)\geq \min_{k\in\mathcal{L}_i} z_k^* \tag{3.29.4}$$

for all $j\in\mathcal{L}$, $i\in\mathcal{S}_j$, where

$$z_j^*=p^1+(c_j/W_j-p^1)W_j/(2\hat{W}_j). \tag{3.29.5}$$

<u>Proof of Lemma 3.2</u>.

We begin by showing that $\liminf_{t\to\infty} \hat{p}_j(t)\geq z_j^*$ and

$\liminf_{t\to\infty} \hat{r}_{ij}(t) \geq \min_{k\in\mathcal{L}_i} z_k^*$. We do this by constructing monotonic increasing sequences of lower bounds $z_j^n \to z_j^*$ for each link j, and a single monotonic increasing sequence of times $\{T^n\}$, such that for all $t \geq T^n$, $\hat{p}_j(t) > z_j^n$ and $\hat{r}_{ij}(t) > \min_{k\in\mathcal{L}_i} z_k^n$, for each $j\in\mathcal{L}$, $i\in\mathcal{S}_j$. We invoke the following lemma.

<u>Lemma 3.2.1</u>. For each $j\in\mathcal{L}$, let $z_j^0=0$ and define

$$z_j^{n+1} = (1-W_j/2\hat{W}_j) \min_{k\in\mathcal{L}} z_k^n + c_j/2\hat{W}_j. \tag{3.30}$$

Then $z_j^n \to z_j^*$ and $z_j^n < c_j/W_j$ for all n.

The proof is given in Appendix B.

Let z_j^n be defined as in (3.30). We now show how to construct the sequence $\{T^n\}$. By Theorem 3.1, we may choose T^0 so that for all $t \geq T^0$

$$\min[t, t-d_{ij}(t)-D_{ik}(t-d_{ij}(t))] > \max_{l\in\mathcal{L}} t_l^0 \tag{3.31}$$

for all $j\in\mathcal{L}$, $i\in\mathcal{S}_j$, $k\in\mathcal{L}_i$. Now if $t \geq T^0$, then $\hat{p}_j(t)=\hat{p}_j^n$ for some $n \geq 1$. Hence, by (3.29.2), $\hat{p}_j(t) > z_j^0 = 0$. Furthermore,

$$\begin{aligned}\hat{r}_{ij}(t) &= \min_{k\in\mathcal{L}_i} \hat{p}_k(t-d_{ij}(t)-D_{ik}(t-d_{ij}(t))) \\ &\geq \min_{k\in\mathcal{L}_i} z_k^0.\end{aligned} \tag{3.32}$$

Now suppose there exists T^n such that for all $t \geq T^n$, $\hat{p}_j(t) > z_j^n$ and $\hat{r}_{ij}(t) > \min_{k\in\mathcal{L}_i} z_k^n$, for all $j\in\mathcal{L}$, $i\in\mathcal{S}_j$. Since $z_j^n < c_j/W_j$, we may choose M large enough such that, for all $m \geq M$, $|e_j^m| < (c_j - W_j z_j^n)/2$ and $t_j^m > T^n$, for each $j\in\mathcal{L}$. Now choose

T^{n+1} such that, for all $t \geq T^{n+1}$,

$$\min[t, t-d_{ij}(t)-D_{ik}(t-d_{ij}(t))] > \max_{l\in\mathcal{L}} t_l^{M}, \tag{3.33}$$

for all $j\in\mathcal{L}$, $i\in\mathcal{S}_j$, $k\in\mathcal{L}_i$. If $t>T^{n+1}$, then $\hat{p}_j(t)=\hat{p}_j^{m+1}$ for some $m\geq M$. Hence,

$$\begin{aligned}
\hat{p}_j^{m+1} &= \max_{i\in\mathcal{S}_j} \hat{r}_{ij}(t_j^m) + (c_j - \sum_{i\in\mathcal{S}_j} \hat{r}_{ij}(t_j^m) - e_j^m)/\hat{W}_j \\
&\geq \max_{i\in\mathcal{S}_j} \hat{r}_{ij}(t_j^m) + (c_j - W_j \max_{i\in\mathcal{S}_j} \hat{r}_{ij}(t_j^m) - e_j^m)/\hat{W}_j \\
&= (1-W_j/\hat{W}_j) \max_{i\in\mathcal{S}_j} \hat{r}_{ij}(t_j^m) + (c_j - e_j^m)/\hat{W}_j \\
&> (1-W_j/\hat{W}_j) \min_{k\in\mathcal{L}} z_k^n + c_j/\hat{W}_j - (c_j - W_j z_j^n)/(2\hat{W}_j) \\
&= (1-W_j/2\hat{W}_j) \min_{k\in\mathcal{L}} z_k^n + c_j/2\hat{W}_j \\
&= z_j^{n+1}.
\end{aligned} \tag{3.34}$$

Thus, $\hat{p}_j(t) > z_j^{n+1}$. Furthermore

$$\begin{aligned}
\hat{r}_{ij}(t) &= \min_{k\in\mathcal{L}_i} \hat{p}_k(t-d_{ij}(t)-D_{ik}(t-d_{ij}(t))) \\
&\geq \min_{k\in\mathcal{L}_i} z_k^{n+1}.
\end{aligned} \tag{3.35}$$

By induction, we have constructed the sequence $\{T^n\}$ as desired.

Next we show that, for each $e>0$, there exists T such that for all $t\geq T$, $\hat{p}_j(t)<p^1+\bar{W}e$ and $\hat{r}_{ij}(t)<p^1+\bar{W}e$ for all $j\in\mathcal{L}^1$, $i\in\mathcal{S}^1$, where $\bar{W}=\max_{j\in\mathcal{L}^1} W_j$. Since we already have $\liminf_{t\to\infty} \hat{p}_j(t)\geq p^1$ and $\liminf_{t\to\infty} \hat{r}_{ij}(t)\geq p^1$ for $j\in\mathcal{L}^1$, $i\in\mathcal{S}^1$, this will complete the proof that $\hat{p}_j(t)\to p^1$ and $\hat{r}_{ij}(t)\to p^1$ for $j\in\mathcal{L}^1$, $i\in\mathcal{S}^1$.

Let j be a link in $\mathcal{L}^1$ and let i be a session on j. Choose N large enough that $\hat{r}_{ij}(t_j^n))>p^1-e$ and $e_j^n<(\overline{W}-W_{j+1})e/2$ for all $n\geq N$. Suppose that $\max_{i\in\mathcal{S}_j}\hat{r}_{ij}(t_j^n)\geq p^1+\overline{W}e$ for some $n\geq N$. Then

$$\begin{aligned}\hat{f}_j^n&\geq(W_{j-1})(p^1-e)+\max_{i\in\mathcal{S}_j}\hat{r}_{ij}(t_j^n)\\&\geq(W_{j-1})(p^1-e)+p^1+\overline{W}e\\&=W_jp^1+(\overline{W}-W_{j+1})e.\end{aligned}\tag{3.36}$$

Hence,

$$\begin{aligned}\hat{p}_j^{n+1}&\leq\max_{i\in\mathcal{S}_j}\hat{r}_{ij}(t_j^n)+(c_j-(W_jp^1+(\overline{W}-W_{j+1})e)-e_j^n)/\hat{W}_j\\&\leq\hat{p}_j^n-(\overline{W}-W_{j+1})e/\hat{W}_j-e_j^n/\hat{W}_j\\&\leq\hat{p}_j^n-(\overline{W}-W_{j+1})e/(2\hat{W}_j)\\&<\hat{p}_j^n,\end{aligned}\tag{3.37}$$

where the first step follows from the update protocol (3.29.1) and the fact that $p^1=c_j/W_j$ for $j\in\mathcal{L}^1$. So as long as $\max_{i\in\mathcal{S}_j}\hat{r}_{ij}(t_j^n)>p^1+\overline{W}e$, $\hat{p}_j^n$ must decrease by at least $(\overline{W}-W_{j+1})e/(2\hat{W}_j)$ until $\max_{i\in\mathcal{S}_j}\hat{r}_{ij}(t_j^n)<p^1+\overline{W}e$ for some t_j^n.

Now suppose $\max_{i\in\mathcal{S}_j}\hat{r}_{ij}(t_j^n)<p^1+\overline{W}e$. Then

$$\begin{aligned}\hat{p}_j^{n+1}&\leq\max_{i\in\mathcal{S}_j}\hat{r}_{ij}(t_j^n)+[c_j-(W_j-1)(p^1-e)-\max_{i\in\mathcal{S}_j}\hat{r}_{ij}(t_j^n)-e_j^n]/\hat{W}_j\\&\leq(1-1/\hat{W}_j)\max_{i\in\mathcal{S}_j}\hat{r}_{ij}(t_j^n)+(p^1+(W_{j-1})e-e_j^n)/\hat{W}_j\\&<(1-1/\hat{W}_j)(p^1+\overline{W}e)+(p^1+(W_{j-1})e+(\overline{W}-W_{j+1})e/2)/\hat{W}_j\\&<(1-1/\hat{W}_j)(p^1+\overline{W}e)+(p^1+(\overline{W}+W_{j-1})e/2)/\hat{W}_j\\&<(1-1/\hat{W}_j)(p^1+\overline{W}e)+(p^1+\overline{W}e)/\hat{W}_j\\&=p^1+\overline{W}e.\end{aligned}\tag{3.38}$$

So once $\max_{i\in\mathcal{S}_j} \hat{r}_{ij}(t_j{}^n)$ falls below $p^1+\overline{We}$, so does $\hat{p}_j{}^{n+1}$, and hence both must remain below $p^1+\overline{We}$.

So for each $e>0$, there exists N such that for all $n\geq N$ $p^1-e<\hat{p}_j(t_j{}^n)<p^1+\overline{We}$ and $p^1-e<\hat{r}_{ij}(t_j{}^n)<p^1+\overline{We}$ for all $j\in\mathcal{L}^1$, $i\in\mathcal{S}^1$. Hence, there must also exist T such that for all $t\geq T$, $p^1-e<\hat{p}_j(t)<p^1+\overline{We}$ and $p^1-e<\hat{r}_{ij}(t)<p^1+\overline{We}$ for all $j\in\mathcal{L}^1$, $i\in\mathcal{S}^1$. This completes the proof of Lemma 3.2.

We are now ready to prove Theorem 3.2.

Proof. Partition the links in $\mathcal{L}$ into sets $\mathcal{L}^1$, $\mathcal{L}^2$, . . . ,$\mathcal{L}^L$, with the property that for each set $\mathcal{L}^k$ there exists p^k such that $p_j{}^*=p^k$ for each $j\in\mathcal{L}^k$ and $p_j{}^*\neq p^k$ for $j\notin\mathcal{L}^k$. Define $\mathcal{S}^k$ as the set of sessions i such that $r_i{}^*=p^k$. Number the sets so that $p^1<p^2<\ldots<p^L$. The proof is by induction on the sets of the partition. First we show that the sequences $\{p_j(n)\}$ for each j in $\mathcal{L}^1$ and $\{r_i(n)\}$ for each i in $\mathcal{S}^1$ converge to p^1. Then we show that if the sequences $\{p_j(n)\}$ for j in $\mathcal{L}^k$ and $\{r_i(n)\}$ for i in $\mathcal{S}^k$ converge to p^k for all $k\leq K$, then the sequences $\{p_j(n)\}$ for j in $\mathcal{L}^{K+1}$ and $\{r_i(n)\}$ for i in $\mathcal{S}^{K+1}$ converge to p^{K+1}.

In order to prove the induction step, we will also need to show that if

$$\liminf_{t\to\infty} r_{ij}(t)>p^K \qquad (3.39)$$

for $j\in\bigcup_{h>K}\mathcal{L}^h$, $i\in(\bigcup_{h>K}\mathcal{S}^h)\cap\mathcal{S}_j$, then (3.39) holds for $j\in\bigcup_{h>K+1}\mathcal{L}^h$, $i\in(\bigcup_{h>K+1}\mathcal{S}^h)\cap\mathcal{S}_j$.

First note that $p^1=\min_{j\in\mathcal{L}} p_j{*}=\min_{j\in\mathcal{L}} c_j/W_j$. Now, let $\hat{\underline{p}}(0)=\underline{p}(0)$ and $\hat{\underline{r}}(0)=\underline{r}(0)$. For each $j\in\mathcal{L}$, let $\hat{W}_j=W_j$ and let $e_j{}^n=0$ for all n. Then $\hat{p}_j(t)=p_j(t)$ and $\hat{r}_{ij}(t)=r_{ij}(t)$ for all t. Therefore, by the lemma, $\lim_{t\to\infty} p_j(t)=p^1=p_j{*}$ and $\lim_{t\to\infty} r_{ij}(t)=p^1=r_i{*}$ for $j\in\mathcal{L}^1$, $i\in\mathcal{S}^1$.

We must also show that (3.39) holds for K=1. Now for $i\in\bigcup_{\ell>1}\mathcal{S}^1$, $r_i{*}>p^1$ and hence $k\notin\mathcal{L}^1$ for each $k\in\mathcal{L}_i$. Therefore

$$\begin{aligned}\liminf_{t\to\infty} r_{ij}(t) &\geq \min_{k\in\mathcal{L}_i} z_k{*}\\ &> p^1, \qquad (3.40.1)\end{aligned}$$

since, by definition, $z_k{*}>p^1$ for $k\notin\mathcal{L}^1$.

Next we show that if the controls and rates converge to p^k for the links and sessions in $\mathcal{L}^k$ and $\mathcal{S}^k$, $k\leq K$, and (3.39) holds for K, then the controls and rates converge to p^{K+1} for the links and sessions in $\mathcal{L}^{K+1}$ and $\mathcal{S}^{K+1}$, and (3.39) holds for K+1.

We begin by defining a new network $\mathcal{N}'=(\mathcal{S}',\mathcal{L}')$ where $\mathcal{L}'=\bigcup_{k>K}\mathcal{L}^k$ and $\mathcal{S}'=\bigcup_{k>K}\mathcal{S}^k$. Assign the links in $\mathcal{L}'$ capacities $c_j'=c_j-\sum_{i\in\mathcal{S}_j\setminus\mathcal{S}_j'} r_i{*}$. Let the weights of the links in $\mathcal{L}'$ be denoted by W_j'. Define $p_j'(t)$, $f_j'(t)$, $r_i'(t)$, $r_{ij}'(t)$, $p_j{}^{n\prime}$ and $f_j{}^{n\prime}$ in the obvious manner. Note that

$p^{K+1}=\min_{j\in\mathcal{L}'} c_j'/W_j'=p^{1'}$. Define $\mathcal{L}^{1'}$ and $\mathcal{S}^{1'}$ in the obvious manner. Note that $\mathcal{L}^{1'}=\mathcal{L}^{K+1}$ and $\mathcal{S}^{1'}=\mathcal{S}^{K+1}$.

For each $j\in\mathcal{L}'$, either $W_j'=0$, or $W_j'>0$. If $W_j'=0$, then each of link j's sessions is in $\mathcal{S}^k$, for some $k\leq K$. So, by the induction hypothesis,

$$\begin{aligned}\lim_{n\to\infty} p_j^{n+1} &= \lim_{n\to\infty} [\max_{i\in\mathcal{S}_j} r_{ij}^n+(c_j-\sum_{i\in\mathcal{S}_j} r_{ij}(t_j^n)/W_j]\\ &= \max_{i\in\mathcal{S}_j} \min_{k\in\mathcal{L}_i} p_k^*+(c_j-\sum_{i\in\mathcal{S}_j} r_i^*)/W_j\\ &=p_j^*. \qquad (3.40.2)\end{aligned}$$

The last step follows from the construction of the fair control vector (3.27.3).

For each $j\in\mathcal{L}'$ such that $W_j'>0$, let $\hat{W}_j'=W_j$ and define

$$\begin{aligned}e_j^n= &\sum_{i\in\mathcal{S}_j\setminus\mathcal{S}_j'} (r_{ij}(t_j^n)-r_i^*)\\ &+W_j(\max_{i\in\mathcal{S}_j'} r_{ij}(t_j^n) - \max_{i\in\mathcal{S}_j} r_{ij}(t_j^n)). \qquad (3.41)\end{aligned}$$

If $\mathcal{S}_j=\mathcal{S}_j'$, the sum is empty and $e_j^n=0$. Let $\hat{p}_j'(0)=p_j'(0)=p_j(0)$ for each $j\in\mathcal{L}'$ and $\hat{r}_i'(0)=r_i'(0)=r_i(0)$ for each $i\in\mathcal{S}'$. Define $\hat{p}_j'(t)$, $\hat{f}_j'(t)$, $\hat{r}_i'(t)$, $\hat{r}_{ij}'(t)$, $\hat{p}_j^{n'}$ and $\hat{f}_j^{n'}$ as described in Lemma 3.2. We will show that $\hat{p}_j^{n'}=p_j^n$ and $\hat{r}_{ij}^{n'}=r_{ij}^n$ for all n. We will then show that (3.29.1) and (3.29.2) are satisfied and that $e_j^n\to 0$. We may then apply the lemma.

We show that $\hat{p}_j^{n'}=p_j^n$ and $\hat{r}_{ij}^{n'}=r_{ij}^n$ by induction. First note that if j is the first link to perform a control update, then for $t \leq t_j^0$, $\hat{p}_k'(t)=p_k(t)$ and $\hat{r}_{ik}'(t)=r_{ik}(t)$ for each $k \in \mathcal{L}'$, $i \in \mathcal{S}_k'$.

Now suppose there exists some t_k^n such that, for $t \leq t_k^n$, $\hat{p}_j'(t)=p_j(t)$ and $\hat{r}_{ij}'(t)=r_{ij}(t)$ for each $j \in \mathcal{L}'$, $i \in \mathcal{S}_j'$. Let t_l^m be the time of the next control update after t_k^n. For $j \neq k$, $\hat{p}_j'(t_l^m) = \hat{p}_j'(t_k^n) = p_j(t_k^n) = p_j(t_l^m)$. Furthermore, for $j=k$,

$$\begin{aligned} \hat{p}_k'(t_l^m) &= \hat{p}_k^{n+1}{}' \\ &= \max_{i \in \mathcal{S}_k'} \hat{r}_{ik}'(t_k^n)+(c_k'-\hat{f}_k^{n}{}'-e_k^n)/W_k' \\ &= \max_{i \in \mathcal{S}_k} r_{ik}(t_k^n)+(c_k-f_k^n)/W_k \\ &= p_k^{n+1} \\ &= p_k(t_l^m). \end{aligned} \qquad (3.42)$$

The second step follows from the defintion of e_k^n. Hence, $\hat{p}_j'(t)=p_j(t)$ for each $j \in \mathcal{L}'$, when $t \leq t_l^m$. This last remark also implies that $\hat{r}_{ij}'(t)=r_{ij}(t)$ for $j \in \mathcal{L}'$, $i \in \mathcal{S}_j'$, when $t \leq t_l^m$.

Because $\hat{r}_{ij}'(t)=r_{ij}(t)$, (3.29.1) is satisfied. Since

$$\begin{aligned} \hat{p}_j^{n+1}{}' &= p_j^{n+1} \\ &\geq \max_{i \in \mathcal{S}_j} r_{ij}(t_j^n)+(c_j-W_j(\max_{i \in \mathcal{S}_j} r_{ij}(t_j^n))/W_j \\ &= c_j/W_j \\ &> 0, \end{aligned} \qquad (3.43)$$

(3.29.1) is also satisfied.

Now we show that $e_j^n \to 0$. By the induction hypothesis $r_{ij}(t)-r_i^* \to 0$ for $j\in\mathcal{L}\setminus\mathcal{L}'$, $i\in\mathcal{S}_j\setminus\mathcal{S}_j'$. Also by hypothesis, $\liminf_{t\to\infty} r_{ij}(t)>p^K$ for $j\in\mathcal{L}'$, $i\in\mathcal{S}_j'$. Hence, there exists some N such that for $n\geq N$

$$r_{ij}(t_j^n)>p^K \tag{3.44}$$

for $j\in\mathcal{L}'$, $i\in\mathcal{S}_j'$. Therefore, for $n\geq N$,

$$\begin{aligned}\max_{i\in\mathcal{S}_j} r_{ij}(t_j^n)>p^K \\ &\geq \max_{i\in\mathcal{S}_j\setminus\mathcal{S}_j'} r_i^* \\ &= \lim_{n\to\infty} \max_{i\in\mathcal{S}_j\setminus\mathcal{S}_j'} r_{ij}(t_j^n))\end{aligned} \tag{3.45}$$

for $j\in\mathcal{L}'$. Thus, for large enough n,

$$\max_{i\in\mathcal{S}_j'} r_{ij}(t_j^n)> \max_{i\in\mathcal{S}_j\setminus\mathcal{S}_j'} r_{ij}(t_j^n) \tag{3.46}$$

and

$$\max_{i\in\mathcal{S}_j} r_{ij}(t_j^n)= \max_{i\in\mathcal{S}_j'} r_{ij}(t_j^n). \tag{3.47}$$

So

$$\lim_{n\to\infty} [\max_{i\in\mathcal{S}_j'} r_{ij}(t_j^n)- \max_{i\in\mathcal{S}_j} r_{ij}(t_j^n)]=0 \tag{3.48}$$

and $e_j^n\to 0$, as desired.

Now, by Lemma 3.2 we have $\hat{p}_j'(t)\to p^{1'}$ for $j\in\mathcal{L}^{1'}$. But $\hat{p}_j'(t)=p_j(t)$ and $\mathcal{L}^{1'}$ is the set of links j with $p_j^*=p^{1'}=p^{K+1}$. So $p_j(t)\to p_j^*$ for $j\in\mathcal{L}^{K+1}$. Similarly, $r_{ij}(t)\to r_i$ for $i\in\mathcal{S}^{K+1}$.

Finally, to complete the induction, we must show that (3.39) holds for K+1. But for $i\in\bigcup_{l>k+1}\mathcal{S}^l$, $r_i^*>p^K$ and hence, $k\notin\bigcup_{l\leq k}\mathcal{L}^l$ for each $k\in\mathcal{L}_i$. Therefore, by the lemma,

$$\begin{aligned}\liminf_{t\to\infty} r_{ij}(t) &= \liminf_{t\to\infty} \hat{r}_{ij}'(t)\\ &\geq \min_{k\in\mathcal{L}_i} z_k{*}\\ &> p^{1'}\\ &= p^{K+1}, \qquad (3.49)\end{aligned}$$

as desired. This completes the proof of Theorem 3.2.

3.5 Asynchronous Algorithms for More General Feasible Sets

In this section we give two asynchronous algorithms that produce fair controls and rates over the more general feasible sets defined by (2.11). The algorithms are obtained by modifying the synchronous algorithm given by (2.2) in much the same way that Hayden's algorithm is modified to give the asynchronous algorithm of the preceeding section.

First we change the asynchronous system model so that

$$r_i(t)=\begin{cases}\min\limits_{j\in\mathcal{L}_i} b_i^{-1}(p_j(t-D_{ij}(t))) & \text{when } t\geq D_{ij}(t)\ \forall j\in\mathcal{L}_i\\ r_i(0) & \text{when } 0\leq t<D_{ij}(t) \text{ for some } j\in\mathcal{L}_i\\ 0 & \text{when } t<0.\end{cases} \qquad (3.50)$$

The new update protocol requires that link j performs updates only when

$$b_i(r_{ij}(t_j^n))\leq p_j(t_j^n) \quad \forall\ i\in\mathcal{S}_j. \qquad (3.51)$$

Unfortunately, when the flows are observed this update protocol is not as easy to implement as the one given by

(3.21). Since the link enforces (3.51) by monitoring its sessions' rates, it must know how to calculate $b_i(r_{ij}(t))$ for each of its sessions. But this defeats the intent of the original algorithm to distribute calculation reasonably between the links and the sessions. When flows are communicated, this is not a problem, since the sessions can simply inform the links of both $r_{ij}(t)$ and $b_i(r_{ij}(t))$.

The update function (3.26) was obtained by replacing all occurences of p_j^n in (1.8.1) with $\max\limits_{i\in\mathcal{S}_j} r_{ij}(t_j^n)$. Hence, (2.2.1) suggests the new update function

$$p_j^{n+1}=(1-a_j^n)\max_{i\in\mathcal{S}_j} b_i(r_{ij}(t_j^n))+a_j^n g_j(c_j-f_j^n) \qquad (3.52.1)$$

where

$$a_j^n=A_j(\max_{i\in\mathcal{S}_j} b_i(r_{ij}(t_j^n))) \qquad (3.52.2)$$

where $A_j(\cdot)$ is some appropriately chosen function. This new update function does produce an algorithm that behaves as desired, but in fact, with the update protocol, it is not necessary to modify the update function of (2.2.1) at all!

To see why this is so, consider the case where $g_j(x)=x$, $b_i(x)=x$ and $a_j^n=1/(W_j+1)$ for each $j\in\mathcal{L}$, $i\in\mathcal{S}$. Then (2.2.1) becomes

$$p_j^{n+1}=p_j^n+(c_j-\bar{f}_j^n-p_j^n)/(W_j+1). \qquad (3.53)$$

This algorithm gives Jaffe's fair rate vector [10]. We interpret this update function as follows. Consider Hayden's algorithm in the case where each link j has a "phantom" session that experiences no feedback or propagation delays, and whose path consists only of link j. Let f_j^n denote the aggregate flow of all other sessions and let W_j be the number of all other sessions on j. Then at all times, each link is controlling at least one of its sessions. But the reason for replacing p_j^n in (1.8.1) with $\max r_{ij}(t_j^n)$ in (3.26) was so that each link could pretend to be controlling one of its sessions, and thereby prevent unlimited increases in its control value. Since each link is already controlling a "phantom" session, it is not necessary to modify the update equation at all.

We give the following theorem.

Theorem 3.3. Let $\mathcal{N}=(\mathcal{J},\mathcal{L})$ be any network. Let $g_j(\cdot)$ and $b_i(\cdot)$ satisfy Assumption 1.1. Let $f_j(t)$, $r_i(t)$, and $r_{ij}(t)$ be given according to (3.7), (3.8) and (3.50). Let $p_j^n=p_j(t_j^n)$ and $f_j^n=f_j(t_j^n)$ where the control update times t_j^n satisfy (3.51), and the delay functions $d_{ij}(t)$ and $D_{ij}(t)$ satisfy (3.10) and (3.11). Let the control update function be given by (2.2.1), with $a_j^n=A_j(p_j^n)$. Let $g_j(x)$, $A_j(x)$, $G_j(x,c,S)$ and $H_j(x,c,S)$ be as in Theorem 2.2. Then, for any initial control vector $\underline{p}^0$ and initial rate vector $\underline{r}^0$, $\lim_{t\to\infty} \underline{p}(t)=\underline{p}^*$ and $\lim_{t\to\infty} \underline{r}(t)=\underline{r}^*$, where $\underline{p}^*$ and $\underline{r}^*$ are the fair allocations over the set defined by (2.12).

Before proceeding with the proof of this theorem, we prove the following useful lemma.

Lemma 3.3. Let $\mathcal{N}=(\mathcal{S},\mathcal{L})$ be any network. Let $b_i(\cdot)$, $g_j(\cdot)$ and $A_j(x)$ be as in Theorem 2.2. For each j, let $\{e_j^n\}$ be any sequence that converges to 0, and let $\{t_j^n\}$ be a sequence that satisfies the conditions given below. Let $\hat{\underline{p}}(t)=\underline{0}$ and $\hat{\underline{r}}(t)=\underline{0}$ for $t<0$ and choose any intial vectors $\hat{\underline{p}}(0)$ and $\hat{\underline{r}}(0)$. Let $\hat{p}_j(t)=\hat{p}_j(0)=\hat{p}_j^0$ for $0<t\le t_j^0$ and define

$$\hat{p}_j^{n+1}=(1-\hat{a}_j^n)p_j^n+\hat{a}_j^n g_j(c_j-\hat{f}_j^n-e_j^n) \tag{3.54}$$

where $\hat{f}_j^n$, $\hat{r}_{ij}(t)$, $\hat{r}_i(t)$ are defined as in (3.7), (3.8) and (3.50), $\hat{a}_j^n=A_j(\hat{p}_j^n)$, and $\hat{p}_j(t)=\hat{p}_j(t_j^n)$ for $t_j^{n-1}<t\le t_j^n$. Assume that $d_{ij}(t)$ and $D_{ij}(t)$ satisfy (3.10) and (3.11). Further suppose that the sequence $\{t_j^n\}$ is chosen so that

$$b_i(\hat{r}_{ij}(t_j^n))\le\hat{p}_j(t_j^n) \qquad \forall i\in\mathcal{S}_j,\quad n\ge0, \tag{3.55.1}$$

and that the sequence $\{e_j^n\}$ is chosen so that p_j^n is bounded below. That is,

$$\hat{p}_j^n>z_j^0 \qquad \forall n\ge1, \forall j\in\mathcal{L}, \tag{3.55.2}$$

for some z_j^0.

Let $p^1=\min_{j\in\mathcal{L}} p_j^*$. Define $\mathcal{L}^1$ and $\mathcal{S}^1$ in the obvious manner. Then $\lim_{t\to\infty}\hat{p}_j(t)=p^1$ and $\lim_{t\to\infty} b_i(\hat{r}_{ij}(t))=p^1$ for all $j\in\mathcal{L}^1$, $i\in\mathcal{S}^1$. Furthermore, $\liminf_{t\to\infty}\hat{p}_j(t)\ge z_j^*$ and $\liminf_{t\to\infty} b_i(\hat{r}_{ij}(t))\ge\min_{k\in\mathcal{L}_i} z_k^*$, for all $j\in\mathcal{L}$, $i\in\mathcal{S}_j$, where $z_j^*=H_j(p^1,c_j,\mathcal{S}_j)$.

Proof of Lemma 3.3.

First, we will show $\liminf_{t\to\infty} \hat{p}_j(t) \ge z_j*$ and $\liminf_{t\to\infty} b_i(\hat{r}_{ij}(t)) \ge \min_{k\in\mathcal{L}_i} z_k*$. Then we show that for each $e>0$, there exists T such that for all $t\ge T$, $p_j(t)<p^1+E_j(e)$ for each $j\in\mathcal{L}^1$, where

$$E_j(e)=2(G_j(p^1-e,c_{j+e},\mathcal{S}_j)-p^1). \qquad (3.56)$$

This is sufficient to show the convergence of the controls, since $X_{j=p^1}$ for $j\in\mathcal{L}^1$, and because $E_j(\cdot)$ has the properties that it is monotonically non-decreasing, and $E_j(0)=0$. The convergence of the rates follows directly from the convergence of the controls.

We show $\liminf_{t\to\infty} \hat{p}_j(t) \ge z_j*$ and $\liminf_{t\to\infty} b_i(\hat{r}_{ij}(t)) \ge \min_{k\in\mathcal{L}_i} z_k*$ by constructing monotonic increasing sequences of lower bounds $z_j^n \to z_j*$ and a single monotonic increasing sequence of times $\{T^n\}$, such that for all $t\ge T^n$, $\hat{p}_j(t)>z_j^n$ and $b_i(\hat{r}_{ij}(t))> \min_{k\in\mathcal{L}_i} z_k^n$, for each $j\in\mathcal{L}$, $i\in\mathcal{S}_j$. Let z_j^0 be as in (3.55.2) and define z_j^n by

$$z_j^{n+1} = H_j(\min_{k\in\mathcal{L}} z_k^n, c_j-|e_j^n|, \hat{\mathcal{S}}_j). \qquad (3.57)$$

First we show how to construct the sequence $\{T^n\}$. Then we show that $z_j^n \to z_j*$.

By Theorem 3.1, we may choose T^0 such that for all $t\ge T^0$

$$\min[t,t-d_{ij}(t)-D_{ik}(t-d_{ij}(t))]> \max_{l\in\mathcal{L}} t_l^0 \qquad (3.58)$$

for all $j\in\mathcal{L}$, $i\in\mathcal{S}_j$, $k\in\mathcal{L}_i$. Now if $t\ge T^0$, then $\hat{p}_j(t)=\hat{p}_j^n$ for some $n\ge 1$. Hence, by (3.55.2), $p_j(t)>z_j^0$. Furthermore,

$$b_i(\hat{r}_{ij}(t))= \min_{k\in\mathcal{L}_i} \hat{p}_k(t-d_{ij}(t)-D_{ik}(t-d_{ij}(t)))$$

$$> \min_{k\in\mathcal{L}_i} z_k^0. \tag{3.59}$$

Now suppose there exists T^n such that for all $t \geq T^n$, $\hat{p}_j(t) > z_j^n$ and $b_i(\hat{r}_{ij}(t)) > \min_{k\in\mathcal{L}_i} z_k^n$, for all $j\in\mathcal{L}$, $i\in\mathcal{S}_j$. Choose M large enough such that, for all $m \geq M$, $|e_j^m| < |e_j^n|$ and $t_j^m > T^n$, for each $j\in\mathcal{L}$. Now choose T^{n+1} such that, for all $t \geq T^{n+1}$,

$$\min[t, t-d_{ij}(t)-D_{ik}(t-d_{ij}(t))] > \max_{l\in\mathcal{L}} t_l^M, \tag{3.60}$$

for all $j\in\mathcal{L}$, $i\in\mathcal{S}_j$, $k\in\mathcal{L}_i$. If $t \geq T^{n+1}$, then $\hat{p}_j(t) = \hat{p}_j^{m+1}$ for some $m \geq M$. The update protocol (3.55.1) guarantees that

$$\hat{f}_j^m \leq \sum_{i\in\mathcal{S}_j} b_i^{-1}(p_j^m). \tag{3.61}$$

Hence, we have

$$\begin{aligned}\hat{p}_j^{m+1} &\geq (1-\hat{a}_j^m)\hat{p}_j^m + \hat{a}_j^m g_j(c_j - \sum_{i\in\mathcal{S}_j} b_i^{-1}(\hat{p}_j^m) - |e_j^n|)\\ &= H_j(\hat{p}_j^m, c_j - |e_j^n|, \mathcal{S}_j)\\ &> H_j(\min_{k\in\mathcal{L}} z_k^n, c_j - |e_j^n|, \mathcal{S}_j)\\ &= z_j^{n+1},\end{aligned} \tag{3.62}$$

where the next to last step follows from (2.27). Thus, $\hat{p}_j(t) > z_j^{n+1}$. Furthermore

$$\begin{aligned}b_i(\hat{r}_{ij}(t)) &= \min_{k\in\mathcal{L}_i} (\hat{p}_k(t-d_{ij}(t)-D_{ik}(t-d_{ij}(t))))\\ &\geq \min_{k\in\mathcal{L}_i} z_k^{n+1}.\end{aligned} \tag{3.63}$$

By induction, we have constructed the sequence $\{T^n\}$ as desired.

Now we show that $z_j^n \to z_j^*$. To do this we invoke the following theorem.

Theorem 3.4. Let S be a linear space with norm $||\cdot||$ such that $\{x: ||x||\leq c\}$ is compact for all c. Let $f:S\to S$ and $f_n:S\to S$ be functions such that $f_n\to f$ uniformly, and such that $||f(x) - f(y)||<||x - y||$ for all $x,y\in S$. Suppose there exists x^* such that $x^*=f(x^*)$. Define $x_{n+1}=f_n(x_n)$. Then $x_n\to x^*$.

The proof of this theorem in given in Appendix C.

Let $F_j^n(\underline{z})=H_j(\min_k z_k, c_j-e_j^n, \mathcal{S}_j)$ and $F_j(\underline{z})=H_j(\min_k z_k, c_j, \mathcal{S}_j)$. Let $F^n(\underline{z})=(\ldots,F_j^n(\underline{z}),\ldots)$ and $F(\underline{z})=(\ldots,F_j(\underline{z}),\ldots)$. Since $e_j(n)\to 0$, $F^n(\underline{z})\to F(\underline{z})$.

Let $||z||=\max_j |z_j|$. Then, since $g_j(\cdot)$ is uniformly continuous, for any $\epsilon>0$, we can find δ such that

$$||F^n(\underline{z})-F(\underline{z})||$$

$$=\max_j |H_j(\min_k z_k, c_j-|e_j^n|, \mathcal{S}_j)-H_j(\min_k z_k, c_j, \mathcal{S}_j)|$$

$$=\max_j A_j(\min_k z_k)$$

$$\cdot|G_j(\min_k z_k, c_j-|e_j^n|, \mathcal{S}_j)-G_j(\min_k z_k, c_j, \mathcal{S}_j)|$$

$$\leq\max_j |g_j(c_j-|e_j(n)|-\sum_{i\in\mathcal{S}_j} b_i^{-1}(\min_k z_k))$$

$$-g_j(c_j-\sum_{i\in\mathcal{S}_j} b_i^{-1}(\min_k z_k))|$$

$$< \delta , \tag{3.64}$$

whenever $|e_j(n)|<\epsilon$. Hence, the convergence of $F_n(\underline{z})$ to $F(\underline{z})$ is uniform.

Furthermore, by (2.27)

$$\begin{aligned}||F(x)-F(y)|| &= \max_j |H_j(\min_k x_k, c_j, \delta_j)-H_j(\min_k y_k, c_j, \delta_j)| \\ &< \max_j |\min_k x_k - \min_k y_k| \\ &= |\min_k x_k - \min_k y_k| \\ &\leq \max_k |x_k - y_k| \\ &= ||\underline{x}-\underline{y}||. \end{aligned} \tag{3.65}$$

Finally we show that z^* is a fixed point of $F(\cdot)$ Let h satisfy $z_h^* = \min_k z_k^*$. Then

$$\begin{aligned} z_h^* &= H_h(z_h^*, c_h, \delta_h) \\ &= X_h, \end{aligned} \tag{3.66}$$

where X_h is defined by (2.6). Now for any j, $z_j^* \geq z_h^* = X_h$ and so

$$\begin{aligned} z_j^* &= H_j(X_h, c_j, \delta_j) \\ &\geq X_h. \end{aligned} \tag{3.67}$$

Therefore

$$\begin{aligned} X_h - H_j(X_h, c_j, \delta_j) &\leq 0 \\ &= X_j - H_j(X_j, c_j, \delta_j), \end{aligned} \tag{3.68}$$

which implies $X_h \leq X_j$. Hence $X_h = \min_k X_k = p^1$ and

$F_j(\underline{z}^*)=H_j(p^1,c_j,\mathcal{S}_j)=\underline{z}^*$.

Hence, by Theorem 2.3, $z^n \rightarrow z^*$.

Next we show that, for each $e>0$, there exists T such that for all $t\geq T$, $\hat{p}_j(t)<p^1+E_j(e)$ and $b_i(\hat{r}_{ij}(t))<p^1+E_j(e)$ for all $j\in\mathcal{L}^1$, $i\in\mathcal{S}^1$.

Let j be a link in $\mathcal{L}^1$ and let i be a session on j. Choose N large enough that $b_i(\hat{r}_{ij}(t_j^n))\geq p^1-e$ and $|e_j^n|<e$ for all $n\geq N$. Now suppose that $p_j^n\geq p^1+E_j(e)$ for some $n\geq N$. Since

$$\hat{f}_j^n>\sum_{i\in\mathcal{S}_j} b_i^{-1}(p^1-e), \tag{3.69}$$

we have

$$\begin{aligned}\hat{p}_j^{n+1} &\leq \hat{p}_j^n+\hat{a}_j^n(g_j(c_j-e_j^n-\sum_{i\in\mathcal{S}_j} b_i^{-1}(p^1-e))-\hat{p}_j^n)\\ &\leq \hat{p}_j^n+\hat{a}_j^n(g_j(c_j+e-\sum_{i\in\mathcal{S}_j} b_i^{-1}(p^1-e))-(p^1+E_j(e)))\\ &=\hat{p}_j^n+\hat{a}_j^n(G_j(p^1-e,c_j+e,\mathcal{S}_j)-(p^1+E_j(e)))\\ &=\hat{p}_j^n-(1/2)A_j(\hat{p}_j^n)E_j(e)\\ &<\hat{p}_j^n.\end{aligned} \tag{3.70}$$

So as long as $\hat{p}_j^n\geq p^1+E_j(e)$, $\hat{p}_j^{n+1}$ will be less than $\hat{p}_j^n$. Furthermore, the amount by which $\hat{p}_j^n$ decreases is at least

$$(1/2)E_j(e)\min\{A_j(x):x\in[p^1+E_j(e),\hat{p}_j^M]\}, \tag{3.71}$$

where M is any time for which $\hat{p}_j^M\geq p^1+E_j(e)$. The minimum in (3.71) must exist because $A_j(x)$ is continuous and positive for all x. Therefore, $\hat{p}_j^n$ must eventually be less than $p^1+E_j(e)$.

Now suppose $\hat{p}_j^n < p^1+F_j(e)$. Then

$$\begin{aligned} \hat{p}_j^{n+1} &\le \hat{p}_j^n + \hat{a}_j^n(G_j(p^1-e, c_j-e_j^n, \mathcal{S}_j)-\hat{p}_j^n) \\ &\le (1-\hat{a}_j^n)\hat{p}_j^n + \hat{a}_j^n G_j(p^1+e, c_j-e, \mathcal{S}_j) \\ &\le (1-\hat{a}_j^n)(p^1+E_j(e)) + \hat{a}_j^n(p^1+(1/2)E_j(e)) \\ &< p^1+E_j(e). \qquad (3.72) \end{aligned}$$

Hence, the controls $\hat{p}_j^n$ converge to p^1, for $j \in \mathcal{L}^1$.

So for each $e>0$, there exists N such that for all $n \ge N$ $p^1-e<\hat{p}_j(t_j^n)<p^1+E_j(e)$ and $p^1-e<\hat{b}_i(r_{ij}(t_j^n))<p^1+E_j(e)$ for all $j \in \mathcal{L}^1$, $i \in \mathcal{S}^1$. Hence, there must also exist T such that for all $t \ge T$, $p^1-e<\hat{p}_j(t)<p^1+E_j(e)$ and $p^1-e<b_i(\hat{r}_{ij}(t))<p^1+E_j(e)$ for all $j \in \mathcal{L}^1$, $i \in \mathcal{S}^1$. This completes the proof of Lemma 3.3.

We are now ready to prove Theorem 3.3.

Proof. Partition the links in $\mathcal{L}$ as described in the proof of Theorem 3.2. The proof is by induction on the sets of the partition, and is analogous to the proof of Theorem 3.2.

Let $\hat{\underline{p}}(0)=\underline{p}(0)$, $\hat{\underline{r}}(0)=\underline{r}(0)$, and $e_j^n=0$ for each $j \in \mathcal{L}$, for all n. Then $\hat{p}_j(t)=p_j(t)$ and $\hat{r}_{ij}(t)=r_{ij}(t)$ for all t. Therefore, by the lemma, $\lim_{t\to\infty} p_j(t)=p^1=p_j^*$ and $\lim_{t\to\infty} b_i(r_{ij}(t))=p^1=r_i^*$ for $j \in \mathcal{L}^1$, $i \in \mathcal{S}^1$.

Next we show that if the controls and rates converge to p^k for the links and sessions in $\mathcal{L}^k$ and $\mathcal{S}^k$, $k \le K$, and (3.72) holds for K, then the controls and rates converge to p^{K+1} for the links and sessions in $\mathcal{L}^{K+1}$ and $\mathcal{S}^{K+1}$, and (3.72) holds for K+1. We begin by defining a new network $\mathcal{N}'=(\mathcal{S}',\mathcal{L}')$ exactly as described in the proof of Theorem 3.2.

For each $j\in\mathcal{L}'$, let

$$e_j^n=\sum_{i\in\mathcal{S}_j\setminus\mathcal{S}_j'}(r_{ij}(t_j^n)-r_i*). \tag{3.73}$$

If $\mathcal{S}_j=\mathcal{S}_j'$, let $e_j^n=0$. By the induction hypothesis, $e_j^n\to 0$. Let $\hat{p}_j'(0)=p_j'(0)=p_j(0)$ for each $j\in\mathcal{L}'$ and $\hat{r}_i'(0)=r_i'(0)=r_i(0)$ for each $i\in\mathcal{S}'$. Define $\hat{p}_j'(t)$, $\hat{f}_j'(t)$, $\hat{r}_i'(t)$, $\hat{r}_{ij}'(t)$, $\hat{p}_j^{n'}$ and $\hat{f}_j^{n'}$ as described in Lemma 3.2. By an argument analogous to that given in the proof of Theorem 3.2, we claim that $\hat{p}_j^{n'}=p_j^n$ and $\hat{r}_{ij}^{n'}=r_{ij}^n$ for all n.

Next we show that (3.54) is satisfied. Because $\hat{r}_{ij}(t)=r_{ij}(t)$, (3.54.1) is satisfied.

Let $z_0<\min(\min_{j\in\mathcal{L}} p_j^0,p^1)$ and let t_1^0 be the time of the first link update. Then $p_j(t)>z_0$ for $t<t_1^0$. Now suppose $p_j(t)>z_0$ for $t<t_j^n$. Then

$$\begin{aligned} p_j^{n+1} &\geq H_j(p_j^n,c_j,\mathcal{S}_j) \\ &>H_j(z_0,c_j,\mathcal{S}_j) \\ &>z_0. \end{aligned} \tag{3.74}$$

The last step follows because $z_0<p^1\leq X_j$. Therefore, $p_j(t)>z_0$ for $t<t_k^m$, where t_k^m is the time of the next link update after t_j^n. So, by induction, $p_j(t)>z_0$ for all t. Hence (3.54.2) is satisfied.

Now, by Lemma 3.2 we have $p_j'(t) \to p^{1'}$ for $j \in \mathcal{L}^{1'}$. But $p_j'(t)=p_j(t)$ and $\mathcal{L}^{1'}$ is the set of links j with $p_j^*=p^{1'}=p^{K+1}$. So $p_j(t) \to p_j^*$ for $j \in \mathcal{L}^{K+1}$. Similarly, $r_{ij}(t) \to r_i^*$ for $i \in \mathcal{S}^{K+1}$. This completes the proof of Theorem 3.3.

A similar theorem can be proved for an algorithm using the update equation (3.52). The proof is essentially the same as for Theorem 3.4, except that in the lemma we define p_j^{n+1} by

$$\hat{p}_j^{n+1}=(1-\hat{a}_j^n)(\max_{i \in \mathcal{S}_j} b_i(\hat{r}_{ij}(t_j^n))+E_j^n) + \hat{a}_j^n g_j(c_j-\hat{f}_j^n-e_j^n) \qquad (3.75)$$

where $\{E_j^n\}$ is any sequence that converges to 0, and $E_j^n \geq 0$ for all n.

Note that Theorem 2.2 follows directly from Theorem 3.4, by letting $d_{ij}(t)=D_{ij}(t)=0$ and choosing $t_j^n=n$ for each $j \in \mathcal{L}$, $i \in \mathcal{S}_j$.

Chapter 4

General Asynchronous Distributed Algorithms

Bertsekas and others [13] have developed some broadly applicable results pertaining to general asynchronous distributed algorithms. In this chapter, we discuss these results and show how they can be modified to include the algorithms presented in Chapter 3.

4.1 A General Convergence Theorem

In this section we present Bertsekas' main result, slightly reformulated to match our model.

For a given feasible set $X \subseteq R^n$, we are interested in finding an element of the solution set $X^* \subset X$. Such an element is called a solution. We consider a system in which a network of N processors iteratively computes estimates of the solution. Each processor i maintains at all times t an estimate of the solution $x_i(t) \in X$, and a vector of m_i measurements $z_i(t)=(z_{i1}(t),\ldots,z_{im_i}(t))$, as communicated to the processor by the network. The estimates and measurements are updated as follows.

Without loss of generality, we index the times at which the events of interest take place (such as a processor updating its estimate or receiving a measurement) by an integer variable t. We also assume that for any integer t only one event of interest occurs in the system. Suppose processor i receives at time t a new value of the measurement $\hat{z}_{ij}(t)$. We call the received value $\hat{z}_{ij}(t)$ to distinguish it from the value $z_{ij}(t)$ currently stored by the processor. Then $z_{ij}(t+1)=\hat{z}_{ij}(t)$. Furthermore, the processor updates its estimate according to

$$x_i(t+1)=M_{ij}(x_i(t),z_{i1}(t),\ldots,\hat{z}_{ij}(t),\ldots,z_{im_i}(t)) \qquad (4.1)$$

where $M_{ij}(\cdot)$ is a given function. We call this a measurement update. If no new measurement is received then $z_{ij}(t+1)=z_{ij}(t)$. Each processor also updates $x_i(t)$ from time to time, according to

$$x_i(t+1)=F_i(x_i(t),z_i(t)) \qquad (4.2)$$

where $F_i(\cdot)$ is a given function. We call this a self-generated update. When no new measurement is received, and the processor does not update its estimate, $x_i(t+1)=x_i(t)$. The measurement $\hat{z}_{ij}(t)$ received by processsor i at time t is related to the processor estimates $x_1,x_2,\ldots,x_N$ by

$$\hat{z}_{ij}(t)=G_{ij}(x_1(T_{ij}{}^1(t)),\ldots,x_N(T_{ij}{}^N(t))). \qquad (4.3)$$

(Bertsekas also includes as an argument to $G_{ij}(\cdot)$ a random variable ω. We omit this for simplicity, since the systems we are studying are deterministic.) We make the following assumptions about the times at which measurements are received or updates take place.

Assumption 4.0. For all $1 \leq i \leq N$, $1 \leq j \leq m_i$, $1 \leq k \leq N$, $T_{ij}^k(t) \leq t$.

Assumption 4.1. If $t_1 \geq t_2$, then

$$T_{ij}^k(t_1) \geq T_{ij}^k(t_2) \tag{4.4}$$

for all $1 \leq i \leq N$, $1 \leq j \leq m_i$, $1 \leq k \leq N$.

Assumption 4.2. For each i and j, and any t_0, there exists $t_1 > t_0$ at which processor i receives a measurement $z_{ij}(t_1)$ of the form (4.3), with $T_{ij}^k(t_1) \geq t_0$ for each $1 \leq k \leq N$. Also, for each i and any t_0, there exists $t_2 > t_0$ at which processor i updates its estimate according to (4.2).

Assumption 4.0 says that delays must be positive, that is, we cannot predict the future. We call this the causality assumption. Assumption 4.1 is essentially equivalent to (3.10). Assumption 4.2 is equivalent to the result of Theorem 3.1, the consequence of (3.10) and (3.11). Bertsekas calls it the continuing update assumption. The new assumptions are required because we no longer assume that the measurements $z_{ij}(t)$ are updated continuously.

Bertsekas gives the following theorem.

Theorem 4.1. Let Assumptions 4.0 - 4.2 hold. Suppose there exists a sequence of sets $\{X(k)\}$ with the following properties:

$$X^* \subset X(k+1) \subset X(k) \subset \ldots \subset X \tag{4.5.1}$$

and

$$X^* = \bigcap_{k=1}^{\infty} X(k). \tag{4.5.2}$$

For $1 \leq i \leq N$, $1 \leq j \leq m_i$, and $k \geq 0$, define

$$Z_{ij}(k)=\{G_{ij}(x_1,\ldots,x_N) \mid x_h \in X(k) \text{ for } 1 \leq h \leq N\} \tag{4.6.1}$$

$$\overline{X}_i(k)=\{F_i(x_i,z_i) \mid x_i \in X(k), z_i \in Z_i(k)\} \tag{4.6.2}$$

$$\overline{Z}_{ij}(k)=\{G_{ij}(x_1,\ldots,x_N) \mid x_h \in \overline{X}_h(k) \text{ for } 1 \leq h \leq N\}, \tag{4.6.3}$$

where $Z_i(k)=Z_{1j}(k) \times \ldots \times Z_{im_i}(k)$. Let $\overline{Z}_i(k)=\overline{Z}_{1j}(k) \times \ldots \times \overline{Z}_{im_i}(k)$.

Suppose that the sets $X(k)$ and the mappings $F_i(\cdot)$, $G_{ij}(\cdot)$ and $M_{ij}(\cdot)$ are such that, for all i, j, k,

$$\overline{X}_i(k) \subset X(k) \tag{4.7.1}$$

$$M_{ij}(x_i,z_i) \in X(k) \quad \text{when } x_i \in X(k),\ z_i \in Z_i(k) \tag{4.7.2}$$

$$M_{ij}(x_i,z_i) \in \overline{X}_i(k) \quad \text{when } x_i \in \overline{X}_i(k),\ z_i \in Z_i(k) \tag{4.7.3}$$

$$M_{ij}(x_i,z_i) \in X(k+1) \quad \text{when } x_i \in \overline{X}_i(k),\ z_i \in \overline{Z}_i(k) \tag{4.7.4}$$

$$F_i(x_i,z_i) \in X(k+1). \quad \text{when } x_i \in X(k+1),\ z_i \in \overline{Z}_i(k) \tag{4.7.5}$$

Then, if all initial processor estimates $x_i(0)$ are in $X(0)$, and all initial measurements $z_i(0)$ are in $Z_i(0)$, the limit points of $\{x_i(t)\}$ are solutions for each $i=1,\ldots,N$.

We interpret the theorem as follows. Condition (4.7) ensures that if all estimates $x_i(t)$ are in $X_i(k)$ and all measurements $z_i(t)$ are in $Z_i(t)$, then eventually, $x_i(t)$ and $z_i(t)$ will enter $X_i(k+1)$ and $Z_i(k+1)$, respectively, and remain there. So for any $k>0$, $x(t) \in X(k)$ for large enough t. Thus, as k increases, $x(t) \in X(k)$ gets arbitrarily close to the solution set. To see how (4.7) guarantees this, consider the following argument.

Let us assume that $x_i(t) \in X_i(k)$ and $z_i(t) \in Z_i(k)$ for all i. Condition (4.7.1) says that after processor i performs a self-generated update, the new estimate will still be in $X_i(k)$. Similarly, (4.7.2) says that after a measurement update, the estimate is still in $X_i(k)$.

After a self-generated update, processor i's estimate is in $\overline{X}_i(k)$. We may regard the membership of $x_i(t)$ in $\overline{X}_i(k)$ as progress toward the goal that $x_i(t)$ eventually be in $X_i(k+1)$. Condition (4.7.3) says that that progress is not undone by any subsequent measurement updates.

Condition (4.7.4) says that after all the processors have made self-generated updates, and enough time has passed for the measurements to relfect this, then processor i's next measurement update will drive $x_i(t)$ into $X_i(k+1)$. Finally, (4.7.5) ensures that after $x_i(t)$ is in $X_i(k+1)$, additional self-generated updates will not push $x_i(t)$ out of $X_i(k+1)$.

Theorem 4.1 is sufficiently general that we are tempted to try to reformulate the asynchronous flow control algorithms in terms that would allow us to apply the theorem. The theorem as given, however, does not apply to the algorithms described in Chapter 3 for two reasons.

Theorem 4.1 states that if an asynchronous algorithm meets certain conditions, the sequence of processor estimates it generates must converge to a solution, regardless of the manner in which the estimates are updated. But the flow control algorithms in Chapter 3 require that updates only occur at specified times, that is, when the update protocol is satisfied.

Furthermore, Theorem 4.1 states that each processor i computes a complete estimate of a solution x*, whereas the flow control algorithms only require each link j to compute an estimate of the jth coordinate of the solution p*.

Both of these difficulties can be overcome by slight reformulations of the theorem, as described in the following sections.

4.2 Algorithms with Partial Processor Estimates

In this section we show how to modify Bertsekas' model to describe algorithms in which each processor estimates only a partial solution. In an earlier paper, Bertsekas [14] gives a result similar to Theorem 4.1 for algorithms where processors compute only partial solutions, but that result is less general than Theorem 4.1, in that the form of the measurements is more constrained.

In general, the dimensionality of the solution may exceed the number of processors. In that case we would need some processors to calculate estimates of several solution coordinates. However, we may consider the n_i coordinates that processor i estimates as an n_i-vector, and then consider that vector as a single coordinate of an N-vector. So, without loss of generality, we assume that each processor i calculates only the ith coordinate of the solution. Indeed, we may consider that in the general algorithm of section 4.1, the limit points that each processor i calculates are the (n-dimensional) ith coordinates of solutions in $(X^*)^N \subseteq X^N \subseteq R^{nN}$.

We may still describe the algorithm using (4.1) - (4.3) by simply reinterpreting $x_i(t)$ as processor i's estimate at t of the ith coordinate of the solution. We write $x(t)=(x_1(t),\ldots,x_N(t))$, where $x_i(t)\in X_i$ for each $i=1,\ldots,N$ and $x(t)\in X=X_1 x \ldots x X_N$. We call $x(t)$ the complete estimate and $x_i(t)$ the ith partial estimate.

We might now state a theorem similar to Theorem 4.1 for this model, but for one remaining difficulty. Even when the processor estimates converge, Theorem 4.1 does not promise that the different processor estimates converge to the same solution. An algorithm where the processors each calculate one coordinate of different solutions is of dubious value. For example, we might have $x_1(t) \rightarrow x_1^* \in X_1^*$ and $x_2(t) \rightarrow x_2^* \in X_2^*$, but $(x_1^*, x_2^*) \notin X^*$. Additional restrictions are needed to guarantee that if the partial estimates converge, they converge to coordinates of the same solution. While more general results may be possible, we choose to avoid the problem by assuming that the solution set X* is the Cartesian product of sets X_i^*, i=1,...,N,

$$X^* = X_1^* x \ldots x X_N^*. \qquad (4.8)$$

Rather than restate Theorem 4.1 for this model, we expand the model in the following section to include update protocols, and so get a more general result.

4.3 General Asynchronous Algorithms with Update Protocols

In Chapter 3, we introduced the idea of an update protocol as a way of restricting when the processors could update their estimates. In this section, we revise our interpretation of an update protocol so that the processors may update at any time, but only when the protocol is satisfied does the update actually affect the estimate.

An update protocol for processor i can be expressed in terms of a protocol function $P_i : X_i x Z_i \rightarrow \{0,1\}$, where Z_i is the set of all possible measurements z_i. If $P_i(x_i, z_i) = 1$, we say that the measurements z_i are consistent with the estimate x_i.

Thus, $P_i(\cdot)$ is the indicator fuction of some subset of $X_i x Z_i$, and we call this set the consistent set C_i. Define

$$\hat{F}_i(x_i,z_i)=\begin{cases} F_i(x_i,z_i) & \text{when } P_i(x_i,z_i)=1 \\ x_i & \text{otherwise} \end{cases} \tag{4.9}$$

and

$$\hat{M}_{ij}(x_i,z_i)=\begin{cases} M_{ij}(x_i,z_i) & \text{when } P_i(x_i,z_i)=1 \\ x_i & \text{otherwise.} \end{cases} \tag{4.10}$$

We call $\hat{F}_i(\cdot)$ and $\hat{M}_{ij}(\cdot)$ the constrained update functions, and $F_i(\cdot)$ and $M_{ij}(\cdot)$ the unconstrained update functions. We rewrite (4.1) and (4.2) as

$$x_i(t+1)=\hat{M}_{ij}(x_i(t),z_{i1}(t),\ldots,\hat{z}_{ij}(t),\ldots,z_{im_i}(t)) \tag{4.11}$$

and

$$x_i(t+1)=\hat{F}_i(x_i(t),z_i(t)). \tag{4.12}$$

Hence, we restrain the processor from changing its estimate unless the processor's current measurements are consistent with its estimate.

We make the following assumption about the update protocol.

Assumption 4.3. For each processor i, there exists an infinite sequence of times for which $P_i(x_i(t),z_i(t))=1$ and the processor updates according to (4.12).

One way to ensure that Assumption 4.3 holds is to require

$$P_i(x_i,(G_{i1}(x^1),\ldots,G_{im_i}(x^{m_i})))=1 \tag{4.13}$$

for all $x^j \in X_1 x \ldots \{x_i\} \ldots x X_N$, $1 \leq j \leq m_i$, for all x_i X_i. Essentially, (4.13) says that if each of the measurements z_{ij} could have been generated using the current estimate x_i, then they are consistent with that estimate.

To see how (4.13) implies Assumption 4.3, suppose there exists some time t_0 such that $P_i(x_i(t), z_i(t))=0$ for all $t \geq t_0$. Then $x_i(t)=x_i(t_0)$ for all $t \geq t_0$. But by Assumption (4.2), there exists $t_1 > t_0$ such that

$$z_{ij}(t_1)=G_{ij}(x_1(T_{ij}{}^1(t)), \ldots, x_i(t_0), \ldots, x_N(T_{ij}{}^N(t))). \qquad (4.14)$$

for each $1 \leq j \leq m_i$. But (4.13) and (4.14) imply

$$P_i(x_i(t_1), z_i(t)_1)=1 \qquad (4.15)$$

which is a contradiction. Hence (4.13) guarantees the existence of an infinite sequence of times for which $P_i(x_i(t), z_i(t))=1$.

We give the following theorem.

Theorem 4.2. Let Assumptions (4.0) - (4.3) hold and let X^* be of the form (4.8). Let the processor estimates $x_i(t)$ be updated according to (4.11) and (4.12). Suppose there exist sequences of sets $\{X_i(k)\}$ for $1 \leq i \leq N$ with the following properties:

$$X^* \subset X(k+1) \subset X(k) \subset \ldots \subset X \qquad (4.16.1)$$

and

$$X^* = \bigcap_{k=1}^{\infty} X(k), \qquad (4.16.2)$$

where $X(k)=X_1(k)\times\ldots\times X_N(k)$.

For $1\leq i\leq N$, $1\leq j\leq m_i$, and $k\geq 0$, define

$$Z_{ij}(k)=\{G_{ij}(x)\mid x\in X(k)\} \tag{4.17.1}$$

$$\overline{X}_i(k)=\{F_i(x_i,z_i)\mid x_i\in X_i(k),z_i\in Z_i(k),P_i(x_i,z_i)=1\} \tag{4.17.2}$$

$$\overline{Z}_{ij}(k)=\{G_{ij}(x)\mid x\in\overline{X}(k)\}, \tag{4.17.3}$$

where $Z_i(k)$ and $\overline{X}(k)$ are defined in the obvious manner.

Suppose that the sets $X(k)$ and the mappings $F_i(\cdot)$, $G_{ij}(\cdot)$ and $M_{ij}(\cdot)$ are such that, for all i, j, k,

$$\overline{X}_i(k)\subseteq X_i(k) \tag{4.18.1}$$

$$M_{ij}(x_i,z_i)\in X_i(k) \quad \text{when } x_i\in X_i(k),\ z_i\in Z_i(k),\ P_i(x_i,z_i)=1 \tag{4.18.2}$$

$$M_{ij}(x_i,z_i)\in X_i(k) \quad \text{when } x_i\in\overline{X}_i(k),\ z_i\in Z_i(k),\ P_i(x_i,z_i)=1 \tag{4.18.3}$$

$$M_{ij}(x_i,z_i)\in\overline{X}_i(k+1) \quad \text{when } x_i\in\overline{X}_i(k),\ z_i\in\overline{Z}_i(k),\ P_i(x_i,z_i)=1 \tag{4.18.4}$$

$$F_i(x_i,z_i)\in X_i(k+1) \quad \text{when } x_i\in X_i(k+1),\ z_i\in\overline{Z}_i(k),\ P_i(x_i,z_i)=1 \tag{4.18.5}$$

Then, if all initial processor estimates $x_i(0)$ are in $X_i(0)$, and all initial measurements $z_i(0)$ are in $Z_i(0)$, the limit points of $\{x(t)\}$ are solutions.

<u>Proof</u>. We will show, by induction, that there exists a monotonic increasing sequence of times $\{t_k\}$ such that

$$x_i(t)\in X_i(k) \qquad \text{for } 1\leq i\leq N \tag{4.19}$$

for all $t\geq t_k$. Therefore, by (4.16) and (4.8), the limit points of $\{x(t)\}$ are solutions.

We begin by showing that

$$x_i(t)\in X_i(0) \qquad\qquad \text{for } 1\leq i\leq N \qquad\qquad (4.20)$$

for all $t\geq t_0=0$. By assumption, (4.20) holds for $t=0$. Now suppose there exists $t'\geq 0$ such that (4.20) holds for all $0\leq t\leq t'$. Our model assumes there is exactly one processor i that performs an update according to either (4.11) or (4.12) at time t'. So for each processor $j=i$, $x_j(t'+1)=x_j(t')$. If $P_i(x_i(t'),z_i(t'))=0$, then $x_i(t'+1)=x_i(t')$ and (4.20) holds for $t'+1$. If $P_i(x_i(t'),z_i(t'))=1$, we consider two cases.

Suppose processor i receives a measurement $\hat{z}_{ij}(t')$ at t' and updates according to (4.11). By the causality assumption, $x_h(T_{ij}{}^h(t'))\in X_i(0)$ for each $1\leq h\leq N$. Thus (4.3) and (4.17.1) imply that $z_{ij}(t')\in Z_{ij}(0)$. Similarly, $z_{ih}(t')\in Z_{ih}(0)$ for $1\leq h\leq m_i$. Hence, by (4.18.2), $x_i(t'+1)\in X_i(0)$.

Now suppose processor i updates according to (4.12) at t'. As above, we argue that $z_{ih}(t')\in Z_{ih}(0)$ for $1\leq h\leq m_i$. Hence, by (4.17.2) and (4.18.1), $x_i(t'+1)\in\overline{X}_i(0)\subseteq X_i(0)$.

Thus, (4.20) is satisfied for $t'+1$, and by induction, (4.20) holds for all $t\geq 0$.

Now suppose there exists k and t_k such that (4.19) holds for all $t\geq t_k$. We will show that there exists $t_{k+1}>t_k$ such that (4.19) holds for all $t\geq t_{k+1}$. We do this by constructing, for each i, an intermediate sequence of times, $t_k<\tau_{ki}<t_{ki}\leq t_k'<\tau_{ki}'<t_{ki}'\leq t_k''$, such that

$$z_i(t)\in Z_i(k) \qquad \text{for } t>\tau_{ki} \qquad (4.21.1)$$

$$x_i(t)\in \overline{X}_i(k) \qquad \text{for } t>t_{ki} \qquad (4.21.2)$$

$$x_j(t)\in \overline{X}_j(k) \qquad \text{for } t>t_k'\text{, for all } j \qquad (4.21.3)$$

$$z_i(t)\in \overline{Z}_i(k) \qquad \text{for } t>\tau_{ki}' \qquad (4.21.4)$$

$$x_i(t)\in X_i(k+1) \qquad \text{for } t>t_{ki}' \qquad (4.21.5)$$

$$x_j(t)\in X_j(k+1) \qquad \text{for } t>t_k''\text{, for all } j \qquad (4.21.6)$$

By the induction hypothesis and the causality assumption, there exists τ_{ki} for each processor i such that (4.21.1) holds.

We claim that (4.21.2) holds when t_{ki} is the time of processor i's first update after τ_{ki} according to (4.12) with $P_i(x_i(t),z_i(t))=1$. Clearly (4.21.2) holds for $t=t_{ki}+1$. Now suppose (4.21.2) holds for some $t\geq t_{ki}+1$. If processor i does not update at t, or $P_i(x_i(t),z_i(t))=0$, then $x_i(t+1)=x_i(t)$ and (4.21.2) holds for t+1. If processor i updates by (4.11) with $P_i(x_i(t),z_i(t))=1$, then by (4.18.3), $x_i(t+1)\in\overline{X}_i(k)$. If processor i updates by (4.12) with $P_i(x_i(t),z_i(t))=1$, then $x_i(t+1)\in\overline{X}_i(k)$ by (4.17.2). Hence, (4.21.2) holds for all $t>t_{ki}$.

Now choose $t_k'=\max\limits_{i} t_{ki}$ and (4.21.3) is satisfied.

By the induction hypothesis and the causality assumption, there exists τ_{ki}' for each processor i such that (4.21.4) holds.

By the continuing update assumption and (4.18.4) there exists t_{ki}' such that (4.21.5) holds for $t=t_{ki}'+1$. Now suppose (4.21.5) holds for some $t \geq t_{ki}'+1$. If processor i does not update at t, or $P_i(x_i(t),z_i(t))=0$, then $x_i(t+1)=x_i(t)$ and (4.21.5) holds for t+1. If processor i updates by (4.11) with $P_i(x_i(t),z_i(t))=1$, then by (4.18.4), $x_i(t+1) \in X_i(k+1)$. If processor i updates by (4.12) with $P_i(x_i(t),z_i(t))=1$, then $x_i(t+1) \in X_i(k+1)$ by (4.18.5). Hence, (4.21.5) holds for all $t > t_{ki}'$.

Hence, we may choose $t_k''=\max_{i} t_{ki}'$ and $t_{k+1}=t_k''+1$. We have constructed the sequence $\{t_k\}$ as desired. This completes the proof of Theorem 4.2.

Note that Theorem 4.1 can be considered a special case of Theorem 4.2, by using the protocol functions $P_i(x_i,z_i)=1$ for all $x_i \in X_i$, $z_i \in Z_i$, and by letting $X_i(k)$ in (4.16) equal $X(k)$ in (4.5), for each i.

While we could use Theorem 4.2 to prove the results of Chapter 3, it is more convenient to apply the following corollary. The corollary is just a simplified form of Theorem 4.2, for the case where there are no measurement updates, only self-generated updates.

Corollary 4.1. Let the processor estimates $x_i(t)$ be updated according to (4.11) and (4.12) with $M_{ij}(x_i,z_i)=x_i$. Let Assumptions 4.0 - 4.3 hold and let X^*, $\{X(k)\}$, $\{\overline{X}(k)\}$, $\{Z_{ij}(k)\}$ and $\{\overline{Z}_{ij}(k)\}$ be as in the statement of Theorem 4.2. Suppose

$$\overline{X}(k)\subseteq X(k+1). \tag{4.22}$$

Then, if $x_i(0)\in X_i(0)$ and $z_i(0)\in Z_i(0)$ for all i, the limit points of $\{x(t)\}$ are solutions.

Proof. We prove the corollary by the application of Theorem 4.2. Since $M_i(x_i,z_i)=x_i$, the conditions (4.18.2) and (4.18.3) are trivially satisfied. The condition (4.22) implies (4.18.4) and (4.18.1). Furthermore, (4.18.1) implies

$$\begin{aligned}\overline{Z}_{ij}(k)&=\{G_{ij}(x)\mid x\in\overline{X}(k)\}\\ &\subseteq\{G_{ij}(x)\mid x\in X(k)\}\\ &=Z_{ij}(k)\end{aligned} \tag{4.23}$$

and so

$$\begin{aligned}&\{F_i(x_i,z_i)\mid x_i\in X_i(k+1),z_i\in\overline{Z}_i(k),P_i(x_i,z_i)=1\}\\ &\qquad\subseteq\{F_i(x_i,z_i)\mid x_i\in X_i(k),z_i\in Z_i(k),P_i(x_i,z_i)=1\}\\ &\qquad=\overline{X}_i(k)\\ &\qquad\subseteq X_i(k+1).\end{aligned} \tag{4.24}$$

Hence (4.18.5) is satisfied.

We have shown that the conditions of Theorem 4.2 are satisfied and therefore, the limit points of $\{x(t)\}$ are solutions. This completes the proof of Corollary 4.1.

In the next section, we use this corollary to obtain a result that shows how some synchronous algorithms can be made to work asynchronously.

4.4 Update Protocols for Synchronous Algorithms

In this section we describe how, starting with a synchronous distributed algorithm taken from a given class of algorithms, we can design an update protocol that allows us to implement the algorithm in an asynchronous manner. Unfortunately, it is not always possible to implement the desired protocol, and so, the result has limited application. We present it mainly for the insight it provides about why such update protocols work.

We model a synchronous algorithm as follows. As in section 4.2, each processor i at time t has an estimate $x_i(t) \in X_i$ of the ith coordinate of a solution $x^* \in X^*$, where X^* is of the form (4.8). All the processors update their estimate simultaneously at each t, according to

$$x_i(t+1)=F_i(x_i(t),z_i(t)) \tag{4.25}$$

where

$$z_{ij}(t)=G_{ij}(x(t)) \tag{4.26.1}$$

and

$$z_i(t)=(\ldots,z_{ij}(t),\ldots). \tag{4.26.2}$$

Note that there are no measurement updates, since all measurements are "received" simultaneously. Combining (4.25) and (4.26) we get

$$\begin{aligned} x(t+1) &= (H_1(x(t)),\ldots,H_N(x(t))) \\ &= H(x(t)) \end{aligned} \qquad (4.27)$$

where $H_i(x)=F_i(x_i,G_i(x))$.

We give the following theorem.

<u>Theorem 4.3</u>. For each i, let $\{X(k)\}$ be a sequence such that

$$X^* \subset X(k+1) \subset X(k) \subset \ldots \subset X \qquad (4.28.1)$$

$$X^* = \bigcap_{k=1}^{\infty} X(k) \qquad (4.28.3)$$

and

$$H(X(k)) \subseteq X(k+1), \qquad (4.28.3)$$

where $X(k)=X_1(k)x\ldots xX_N(k)$ and $H(X(k))$ is the image of $X(k)$ under the mapping $H(\cdot)$.

Let $\{x(t)\}$ be generated by (4.3) and (4.9) - (4.12) with $x(0)\in X(0)$ and $z_i(0)\in Z_i(0)$ for each i. Let Assumptions 4.1 and 4.2 hold. If it is possible to define $P_i(x_i,z_i)$ such that

$$\begin{aligned} &\{F_i(x_i,z_i) \mid x_i \in X_i(k), z_i \in Z_i(k), P_i(x_i,z_i)=1\} \\ &\qquad \subseteq \{F_i(x_i,G_i(x))) \mid x \ X(k)\}, \end{aligned} \qquad (4.29)$$

and such that Assumption 4.3 holds, then the limit points of $\{x(t)\}$ are solutions.

<u>Proof</u>. We prove Theorem 4.3 by the application of Corollary

4.1. Since there are no measurement updates, we take $M_{ij}(x_i,z_i)=x_i$. Combining (4.17.1), (4.17.2) and (4.29) we get

$$\begin{aligned}\overline{X}_i(k)&=\{F_i(x_i,z_i)\mid x_i\in X_i(k),z_i\in Z_i(k),P_i(x_i,z_i)=1\}\\ &\subseteq\{F_i(x_i,G_i(x))\mid x\in X(k)\}\\ &=\{H_i(x)\mid x\in X(k)\}\\ &=H_i(X(k))\\ &\subseteq X_i(k+1).\end{aligned} \tag{4.30}$$

Hence, (4.22) holds and the conditions of Corollary 4.2 are satisfied. This completes the proof Theorem 4.3.

The first part of the theorem simply states conditions on the manner of convergence of the estimates generated by the synchronous algorithm. Clearly, if $x(0)\in X(0)$, the sequence $\{x(t)\}$ generated by the synchronous algorithm (4.27) is such that $x(k)\in X(k)$ for all k, and hence its limit points are solutions.

Now consider an asynchronous algorithm that satisfies Corollary 4.1. The estimate sequence $\{x_i(t)\}$ behaves as desired because (4.22) guarantees that, with $x_i(t)\in X_i(k)$ and $z_i(t)\in Z_i(k)$, updating will never cause $x_i(t)$ to go back to $X_i(k-1)$, and because Assumption 4.3 guarantees that $x_i(t)$ will eventually enter $X_i(k+1)$. Now take an algorithm for which (4.22) does not hold. If we choose a protocol such that updating is forbidden whenever updating would cause $x_i(t)$ to go back to $X_i(k-1)$, then (4.22) will be satisfied.

Equation (4.29) tells us how to design that update protocol. We do this by comparing the image of $X_i(k)xZ_i(k)$ under $F_i(\cdot)$ with the image of $X_i(k)xG_i(X(k))$ under $F_i(\cdot)$. The set $X_i(k)xG_i(X(k))$ consists of all estimate and measurement pairs for processor i such that each of the measurements could have been generated from the same element $x\in X(k)$. Hence, $X_i(k)xG_i(X(k))$ is a subset of $X_i(k)xZ_i(k)$. Now choose the consistent set C_i such that $F_i(\cdot)$ maps $X_i(k)xZ_i(k)\cap C_i$ to the image of $X_i(k)xG_i(X(k))$ under $F_i(\cdot)$. Then, with the protocol function equal to the indicator function on C_i, (4.29) holds. The idea behind (4.29) is that updates are permitted only when updating will not push $x_i(t)\in X_i(k)$ back into $X_i(k-1)$.

While we can always select the consistent set so that (4.29) holds for any given k, it may not be possible to choose the set so that (4.29) holds for all k. Even if (4.29) holds for all k, the update protocol may not satisfy Assumption 4.3. Furthermore, it may not be possible to implement the resulting update protocol, since doing so might require the processors to know the exact form of the measurement generation functions $G_{ij}(\cdot)$.

Though these limitations restrict the usefulness of Theorem 4.3, it still provides a starting point for someone trying to design an update protocol.

4.5 The Flow Control Algorithm as an Example

In this section, we give an alternate proof of Theorem 3.2, using Corollary 4.1. In fact, we prove a somewhat stronger result than Theorem 3.2, since we will show that the update protocol is not needed for the controls to converge. We note, however, than in order to apply Corollary 4.1, we must make the causality assumption, which was not required for the proof of Theorem 3.2. The update protocol is required, even with causality, for the algorithm of Theorem 3.3, as a simple example will show.

We begin by showing how the elements of the flow control model fit the general model of sections 4.2 and 4.3. The feasible set X is taken to be R^N, where N is the number of links in the network, and the solution set is $X^*=\{p^*\}$, where p^* is the fair allocation over Hayden's feasible set (3.27.1). The processors are the links and the estimates they compute are the coordinates of the control vector p. The measurements z_{ji} received by link j are just the rates r_{ij} of its sessions. Since the links receive new measurements continuously, but do not update each time new measurements are received, the only events of interest are the control updates which take place at times t_j^n, for $j\in\mathcal{L}$, $n\geq 0$. In the notation of Chapter 4, we have

$$M_{ij}(p_j,r_j)=p_j \tag{4.31.1}$$

$$F_j(p_j,r_j)=\max_{i\in\mathcal{S}_j} r_{ij}+(c_j-\sum_{i\in\mathcal{S}_j} r_{ij})/W_j \tag{4.31.2}$$

and

$$G_{ij}(p)=\min_{k\in\mathcal{L}_i} p_k \qquad \text{for } i\in\mathcal{S}_j. \tag{4.31.3}$$

The original protocol function is given by

$$P_j(p_j,r_j)=\begin{cases}1 & \text{if } \max\limits_{i\in\mathcal{S}_j} r_{ij}\leq p_j\\ 0 & \text{otherwise.}\end{cases} \tag{4.32}$$

However, we will show that the conditions of Corollary 4.1 can be met using $P_j(p_j,r_j)=1$ for all $p_j,r_j\in R$.

To apply the corollary, we must construct the sequences $\{X_j(n)\}$ such that

$$X_j(n+1)\leq X_j(n)\subseteq \ldots \subseteq X_j(0). \tag{4.33.1}$$

$$\bigcap_{n=1}^{\infty} X_j(n)=\{p_j*\} \tag{4.33.2}$$

and

$$F_j(p_j,r_j)\in X_j(n+1) \tag{4.33.3}$$

when $p_j\in X_j(n)$, $r_j\in Z_j(n)$.

Even though we have shown in Chapter 3 that the controls of the flow control algorithm converge for any initial controls $p_j(0)\in R$ and rates $r_{ij}(0)\in R$, there exist networks for which it is not possible to construct a chain of sets $X(n)$ as in (4.33) with $X(0)=R^N$. This is because, with $X(0)=R^N$, $\overline{X}(0)=R^N$ which implies $X(1)=R^N$. So, by induction $X(n)=R^N$ for all n. Instead, for any initial $p_j(0)$ and $r_{ij}(0)$, we take

$$X_j(0)=[-A_j,B_j], \tag{4.34}$$

for suitably large A_j, B_j.

Now partition the links $\mathcal{L}$ into sets $\mathcal{L}^1,\ldots,\mathcal{L}^L$ and the sessions $\mathcal{S}$ into sets $\mathcal{S}^1,\ldots,\mathcal{S}^L$ as in the proof of Theorem 3.2. We will construct $\{X(n)\}$ by finding sequences $\{e^k(n)\}$ and $\{E^k(n)\}$ for each $k=1,\ldots,L$, such that

$$X_j(n)=[p^k-e^k(n),p^k+E^k(n)] \qquad \text{for } j\in\mathcal{L}^k \tag{4.35}$$

and $\{X_j(n)\}$ satisfies (4.33).

Before defining $\{e^k(n)\}$ and $\{E^k(n)\}$, we describe the idea behind their construction. We begin by taking $e^1(n)=0$ for $n\geq 1$ and finding $\{E^1(n)\}$ that is monotonically decreasing and converges to 0. As long as $E^1(n)$ is greater than some threshold less than p^2, we keep $E^k(n)$ fixed at some large number E and let $p^k-e^k(n)=p^1-e^1(n)$, for all $k\geq 2$. When $E^1(n)$ drops below the threshold at some time N_2, we let $p^2-e^2(n)$ rise above p^1 and $E^2(n)$ begins to fall. We still keep $E^k(n)$ fixed at E and let $p^k-e^k(n)=p^2-e^2(n)$ for $k\geq 3$. This process is repeated until $\{e^k(n)\}$ and $\{E^k(n)\}$ have been found for all k.

We formally construct the sequences $\{e^k(n)\}$, $\{E^k(n)\}$ as follows. First define, for $k=1,\ldots,L$,

$$e^k(0)= aE \tag{4.36.1}$$

and

$$E^k(0)= E \tag{4.36.2}$$

where $a= \min 1/W_j$ and E is some suitably large constant such that $E\geq p^L/a$. Now let

$$e^1(n)=0 \tag{4.37.1}$$

and

$$E^1(n+1)=(1-a)E \qquad (4.37.2)$$

for all $n\geq 1$. Suppose that we are given a sequence $N_1<N_2<\ldots<N_L$ such that $N_1=1$ and, for $k>1$,

$$e^{k-1}(n)+E^{k-1}(n)<(p^k-p^{k-1})a/(1-a) \qquad (4.38)$$

for all $n\geq N_k$. Now define, for $n\geq N_k$

$$e^k(n+1)=(1-a)(e^k(n)+E^{k-1}(n)) \qquad (4.39.1)$$

$$E^k(n+1)=(1-a)(e^k(n)+E^k(n)) \qquad (4.39.2)$$

and for $N_k\leq n<N_{k+1}$, $K>k$,

$$e^K(n+1)= p^K-p^k+e^k(n+1) \qquad (4.40.1)$$

$$E^K(n+1)=E. \qquad (4.40.2)$$

In Appendix D, we show by induction that the sequence $N_1<N_2<\ldots<N_L$ satisfying (4.38) exists, and that the sequences $\{e^k(n)\}$ and $\{E^k(n)\}$ are monotonically non-increasing and converge to 0, for all k. Hence, (4.33.1) and (4.33.2) are satisfied.

Furthermore, we show that the sequences have the properties that for $k<K$, for all n,

$$p^k-e^k(n)\leq p^K-e^K(n) \qquad (4.41.1)$$

$$e^k(n)\leq e^K(n) \qquad (4.41.2)$$

and

$$E^k(n)\leq E^K(n). \qquad (4.41.3)$$

These properties will be needed to show that (4.33.3) holds.

We show that (4.33.3) is satisfied for n=0. Note that

$$(W_j-1)(p^k-aX)+\max_{i\in \mathcal{S}_j} r_{ij}\leq \sum_{i\in \mathcal{S}_j} r_{ij} \leq W_j \max_{i\in \mathcal{S}_j} r_{ij}, \qquad (4.42)$$

for $r_j \in Z_j(0)$. Therefore,

$$\begin{aligned} F_j(p_j,r_j) &\geq \max_{i\in\mathcal{S}_j} r_{ij}+(c_j-W_j \max_{i\in\mathcal{S}_j} r_{ij})/W_j \\ &\geq c_j/W_j \\ &\geq p^1 \end{aligned} \tag{4.43}$$

Also,

$$\begin{aligned} F_j(p_j,r_j) &\leq \max_{i\in\mathcal{S}_j} r_{ij}+(c_j-(W_j-1)(p^k-aX)- \max_{i\in\mathcal{S}_j} r_{ij})/W_j \\ &\leq (1-1/W_j)(\max_{i\in\mathcal{S}_j} r_{ij}-p^k+aX)+c_j/W_j \\ &\leq (1-1/W_j)(E+aE)+c_j/W_j \\ &\leq (1-a)(1+a)E+p^k \\ &< p^k+E. \end{aligned} \tag{4.44}$$

Hence,

$$F_j(p_j,r_j)\in[p^1,p^k+E]=X_j(1), \tag{4.45}$$

as desired.

Before showing that (4.33) holds for $n\geq 1$, we make the following observations. If session i on link j is in $\mathcal{S}^k$, then every link in its path is in $\mathcal{L}^K$ for some $K\geq k$. Thus, (4.31.3) and (4.41.1) imply

$$\begin{aligned} r_{ij} &\geq \min_{K>k} p^K-e^K(n) \\ &= p^k-e^k(n) \end{aligned} \tag{4.46}$$

for $i\in\mathcal{S}^k$, $r_{ij}\in Z_{ij}(n)$. Furthermore,

$$r_{ij}\leq p^k+E^k(n) \tag{4.47}$$

for $i\in\mathcal{S}^k$, $r_{ij}\in Z_{ij}(n)$.

Now we show that (4.33) holds for $n\geq 1$. First we show that

$$F_j(p_j,r_j)\leq p^K+E^K(n+1) \tag{4.48}$$

for $j\in\mathcal{L}^K$, $p_j\in X_j(n)$, $r_j\in Z_j(n)$. We consider two cases: $n<N_K$ and $n\geq N_K$. If $n<N_K$, by (4.44)

$$F_j(p_j,r_j)<p^K+E=p^K+E^K(n+1). \tag{4.49}$$

Now suppose $n\geq N_K$. Let $W_j{}^k$ be the number of sessions on link j in $\mathcal{S}^k$. Then, by (4.46)

$$\begin{aligned} r_{ij} &\geq \sum_{k=1}^{K} W_j{}^k(p^k-e^k(n))+\max_{i\in\mathcal{S}_j} r_{ij}-(p^k-e^k(n)) \\ &=c_j-\sum_{k=1}^{K} W_j{}^k e^k(n) +\max_{i\in\mathcal{S}_j} r_{ij}-(p^k-e^k(n)). \end{aligned} \tag{4.50}$$

Therefore,

$$\begin{aligned} F_j(p_j,r_j) &\leq \max_{i\in\mathcal{S}_j} r_{ij}+ [\sum_{k=1}^{K} W_j{}^k e^k(n)- \max_{i\in\mathcal{S}_j} r_{ij}+p^K-e^K(n)]/W_j \\ &=(1-1/W_j)\max_{i\in\mathcal{S}_j} r_{ij}+[\sum_{k=1}^{K} W_j{}^k e^k(n)+p^K-e^K(n)]/W_j \\ &\leq(1-1/W_j)(p^K+E^K(n))+e^K(n)+[p^K-e^K(n)]/W_j \\ &=p^K+(1-1/W_j)(E^K(n))+e^K(n)) \\ &\leq p^K+(1-a)(E^K(n)+e^K(n)) \\ &=p^K+E^K(n+1), \end{aligned} \tag{4.50}$$

as desired. The third step follows from (4.47) and (4.41.2).

Next we show that

$$F_j(p_j,r_j)\geq p^K-e^K(n+1) \tag{4.52}$$

for $j\in\mathcal{L}^K$, $p_j\in X_j(n)$, $r_j\in Z_j(n)$.

First we derive some preliminary results. For convenience, define

$$\overline{W}_j{}^K=W_j-\sum_{k=1}^{K} W_j{}^k. \tag{4.53.1}$$

and

$$a_j^K=\overline{W}_j^K/W_j. \tag{4.53.2}$$

Then $\overline{W}_j^K$ is the number of sessions on link j that are controlled by links at a higher level than K, that is, the number of sessions that are in $\bigcup_{k>K}\mathcal{S}^k$.

Let J be the largest number for which that $i\in\mathcal{S}^J$, $i\in\mathcal{L}_j$. For any $H\leq J$, (4.47) implies

$$\sum_{i\in\mathcal{S}_j} r_{ij}\leq\sum_{k=1}^{H-1} W_j^k(p^k+E^k(n))+\overline{W}_j^{H-1}\max_{i\in\mathcal{S}_j} r_{ij}. \tag{4.54}$$

Therefore

$$\begin{aligned}F_j(p_j,r_j)&\geq\max r_{ij}+[c_j-\sum_{k=1}^{H-1} W_j^k(p^k+E^k(n))-\overline{W}_j^{H-1}\max_{i\in\mathcal{S}_j} r_{ij}]/W_j\\&\geq(1-a_j^{H-1})\max_{i\in\mathcal{S}_j} r_{ij}\\&\quad+a_j^{H-1}(c_j-\sum_{k=1}^{H-1} W_j^k p^k)/\overline{W}_j^{H-1}-\sum_{k=1}^{H-1} a_j^k E^k.\end{aligned} \tag{4.55}$$

By the construction of the fair control vector

$$p^H\leq(c_j-\sum_{k=1}^{H-1} W_j^k p^k)/\overline{W}_j^{H-1}. \tag{4.56}$$

By (4.41.3)

$$\begin{aligned}\sum_{k=1}^{H-1} a_j^k E^k(n)&\leq\sum_{k=1}^{H-1} a_j^k E^{H-1}(n)\\&=(1-a_j^{H-1})E^{H-1}(n),\end{aligned} \tag{4.57}$$

and by the choice of J and (4.41.1)

$$\begin{aligned}\max_{i\in\mathcal{S}_j} r_{ij}&\geq p^J-e^J(n)\\&\geq p^H-e^H(n).\end{aligned} \tag{4.58}$$

Hence,

$$F_j(p_j,r_j)\geq(1-a_j^{H-1})(p^H-e^H(n)-E^{H-1}(n))+a_j^{H-1}p^H$$

$$=p^{H}-(1-a_{j}{}^{H-1})(e^{H}+E^{H-1}(n))$$
$$\geq p^{H}-(1-a)(e^{H}(n)+E^{H-1}(n)). \qquad (4.59)$$

We are ready to show that (4.52) holds. We consider two cases: J=K and J<K. Suppose J=K. If $n \geq N_K$, then (4.59) implies

$$F_j(p_j,r_j)\geq p^{K}-(1-a)(e^{K}(n)+E^{K-1}(n))$$
$$=p^{K}-e^{K}(n+1). \qquad (4.60)$$

If $n<N_K$, let H be the largest number such that $n \geq N_H$. Then

$$F_j(p_j,r_j)\geq p^{H}-e^{H}(n+1),$$
$$=p^{K}-e^{K}(n+1). \qquad (4.61)$$

Now suppose J<K. Then (4.61) also holds for $n<N_{J+1}$. Finally, suppose $n \geq N_{J+1}$. By (4.47)

$$\sum_{i\in S_j} r_{ij}\leq \sum_{k=1}^{J} W_j{}^{k}(p^k+E^k(n))+\max_{i\in S_j} r_{ij}-(p^J+E^J(n)). \qquad (4.62)$$

Thus,

$$F_j(p_{kj},r_j)\geq(1-1/W_j)\max_{i\in S_j} r_{ij}$$
$$+(c_j-\sum_{k=1}^{J} W_j{}^{k}(p^k+E^k(n))-p^J-E^J(n))/W_j$$
$$\geq(1-1/W_j)(\max_{i\in S_j} r_{ij}-E^J(n))-p^J/W_j+(c_j-\sum_{k=1}^{J} W_j{}^{k}p^k)/W_j$$
$$\geq(1-1/W_j)(p^J-e^J(n)-E^J(n))-p^J/W_j+(c_j-\sum_{k=1}^{J} W_j{}^{k}p^k)/W_j$$
$$\geq p^J+(c_j-\sum_{k=1}^{J} W_j{}^{k}p^k)/W_j-(1-a)(e^J(n)+E^J(n))$$
$$=p^K-(1-a)(e^J(n)+E^J(n)), \qquad (4.63)$$

where the last step follows from the construction of the fair control vector. But, since $n \geq N_{J+1}$,

$$\begin{aligned}(1-a)(e^J(n)+E^J(n)) &\leq (1-a)(e^{J+1}(n)+E^J(n)) \\ &= e^{J+1}(n+1) \\ &\leq e^K(n+1) \qquad (4.64)\end{aligned}$$

and so

$$F_j(p_j,r_j) \geq p^K-e^K(n+1). \qquad (4.65)$$

We have shown that,

$$\begin{aligned}F_j(p_j,r_j) &\in [p^K-e^K(n+1),p^K+E^K(n+1)] \\ &= X_j(n+1) \qquad (4.66)\end{aligned}$$

for $j \in \mathcal{L}^K$, $p_j \in X_j(n)$, $r_j \in Z_j(n)$. The conditions of Corollary 4.1 are satisfied, and therefore, the controls generated by the asynchronous flow control algorithm of Theorem 3.2 converge to p* without the update protocol.

It is instructive to consider why the proof in Chapter 3 requires the update protocol while this proof does not. In both cases we must show that, for any e>0, the control $p_j(t)$ eventually drops below p_j^*+e. A necessary step in showing this is demonstrating that

$$p_j^{n+1} \leq p_j^{n-m}+(c_j-f_j^n)/W_j \qquad (4.67)$$

for some non-negative m. With the causality assumption,

$$\max_{i \in \mathcal{S}_j} r_{ij}(t_j^n) \leq p_j^{n-m} \qquad (4.68)$$

for some $m \geq 0$, and so (4.67) is satisfied. With the update protocol, (4.68) holds for m=0 and the causality assumption is not needed.

This leads us to question whether the update protocol is required for the generalized algorithm, as given in Theorem 3.3, if we assume causality. In fact, the protocol is needed, as the following example shows.

Consider a network of one link with capacity c and W sessions. Let $g(x)=x$ and $b_i(x)=x$, for all i. Then

$$p(n+1)=(1-a)p(n)+a(c-f(n)), \tag{4.69}$$

where $a=1/(W+1)$. Note that $p^*=ac$. Now suppose that $f(0)=c$, and the link updates many times in rapid succession, so that f(0) does not have time to change from one update to the next. Then, after N_1 updates,

$$\begin{aligned} p(N_1)&=(1-a)^{N1}p(0)+(1-a^{N1})(c-f(0)) \\ &=c-f(0)-e(N_1). \end{aligned} \tag{4.70}$$

We can make $e(N_1)$ as small as we like by taking large enough N_1. Now the link waits until the flow reflects this control, that is,

$$f(N_1)=W(c-f(0)-e(N_1)). \tag{4.71}$$

Once again, the link updates rapidly, without waiting for the flow to change, so that

$$\begin{aligned} p(N_2)&=c-f(N_1)-e(N_2) \\ &=c-W(c-f(0)-e(N_1))-e(N_2). \end{aligned} \tag{4.72}$$

In this manner, we can construct a sequence of times $\{N_k\}$ such that the flows are given by

$$f(N_{k+1})=W(c-W(c-f(N_k)-e(N_k))-e(N_{k+1})). \qquad (4.73)$$

For appropriately chosen $\{e(N_k)\}$, this sequence of flows will never converge.

Let us see where the conditions of Corollary 4.1 fail to hold for this example. Suppose we have $X(n)=[p^*-e(n),p^*+E(n)]$. Then $Z_i(n)=X(n)$ for each i. Now suppose that $p(n)=p^*-e(n)$ and $r_i(n)=p^*+E(n)$. Then

$$\begin{aligned} p(n+1)&=(1-a)(p^*-e(n))+a(c-W(p^*+E(n)) \\ &=p^*-(1-a)(e(n)+E(n)). \qquad (4.74) \end{aligned}$$

Hence, we cannot guarantee that $F(p,r)\in X(n)$ when $p\in X(n)$, $r\ Z(n)$. Thus, for this algorithm, the update protocol is required to ensure that the controls get above a given threshold. Examining the proof of Theorem 3.3, we see that the update protocol was invoked for that very reason.

We conclude this chapter by remarking that these results are not obvious. Without applying Theorem 4.2 to the flow control algorithms, we would probably have never discovered that one algorithm requires the update protocol and the other does not.

Chapter 5

Simulation and Results

In this chapter we describe a computer program designed to simulate a voice packet network using a flow control algorithm, such as one of those described in Chapter 3. The program was written in Lisp on the Symbolics 3600, by Allan Wechsler and myself. The program listing is given in Appendix E. After describing the program model and the program, we introduce a specific network model and discuss the results of the simulation using that model.

5.1 The Simulation Model

The simulation model is substantially the same as that used by Hayden [9], with some minor differences.

The program allows the user to define a network with an arbitrary topology. The network is specified by two global variables: a list of its users (sessions) and a list of its links.

A user is a data object that has, among other attributes, a rate, a partner (the other user that it talks to) and a route (the list of the links that are in its path). If user i's partner is user k, then user k's partner is user i. At any given time a user is either active (talking) and its partner is inactive (silent), or vice versa.

When a user is active, it generates a variable length voice packet approximately every 20 msec. The actual time between packet generations is a random variable uniformly distributed between 18 and 22 msec. The length of a voice packet in bits is given by

$$\text{PACKET-LENGTH}=\lceil \text{USER-RATE}/50 \rceil . \tag{5.1}$$

The rate is a floating point number, but the length is an integer.

After generating a voice packet, an active user will generate another voice packet with probability (1-p). Otherwise, the packet generated is the last in its talk-spurt. If the packet is designated last, the user becomes inactive after the packet is transmitted and the partner becomes active when the packet is received. We use p=1/60, and hence, the number of packets per talk-spurt is a geometric random variable with mean 60. This conforms with experimental values measured by Brady [15], who found actual talk-spurt durations to be approximately exponentially distributed, with mean 1.2 sec. This is not a completely accurate representation of real speech, however, since brief periods of silence usually occur between talk-spurts, and talk-spurts do not always alternate strictly between two members of a conversation. Nevertheless, it is an acceptable approximation for our purposes. We also have the option of setting the talk-spurt length to infinity. This lets us study the steady state behavior of the system.

When a user is inactive, it generates fixed length control packets at regular intervals, for the purpose of passing feedback information to its partner. The length of the control packets is 10 bits, and the time between the generation of successive control packets is 100 msec.

A link is a data object that has a control value, a flow, a list of its currently active users and a queue of packets waiting to be transmitted on the link.

A link's list of its active users also includes, for each user, the user's rate, as determined by examining the most recently received packet from that user. This rate list is used to determine whether the update protocol is satisfied, and to calculate the flow of the link. This is the main difference between our simulation and Hayden's. In Hayden's program, a link determines its flow by observing the number of bits arriving in the queue over a given period of time. This affects the value of the flow in three ways not accounted for by the theoretical model. First, the observed flow includes control packet traffic. Also, the observed flow for a link is limited by the capacity of the neighboring links that feed its queue. Finally, the observed flow may represent the rates of the link's different users in unequal proportions. For these reasons, we prefer to calculate a link's flow by summing the rates of the link's active users.

Each link attempts to update its control periodically, by first checking whether the specified update protocol is satisfied. If it is, the link updates according to the specified update function. If not, the link waits a given interval and tries again. There are two update interval parameters that can be adjusted: the time between a succesful update and the next update attempt, and the time between an unsuccessful attempt and the next attempt. They are called the UPDATE-INTERVAL and the UPDATE-ATTEMPT-INTERVAL. In section 5.4 we will see that the choice of these parameters is critical to the performance of the system.

One of two update protocols can be selected: MOSELY or HAYDEN. The MOSELY protocol is given by (3.21) and the HAYDEN protocol always permits updates.

Three different update functions can be selected. The HAYDEN-UPDATE-FUNCTION is given by

$$p_j(t+1)=\max[c_j/W_j,\min[c_j,p_j(t)+(c_j-f_j(t))/W_j]]. \qquad (5.2.1)$$

The MOSELY-UPDATE-FUNCTION is given by

$$p_j(t+1)=\max[c_j/W_j,$$
$$\min[c_j,\max_{i\in S_j} r_{ij}(t)+(c_j-f_j(t))/W_j]]. \qquad (5.2.2)$$

The JAFFE-UPDATE-FUNCTION is given by

$$p_j(t+1)=\max[c_j/(W_j+1),$$
$$\min[c_j,p_j(t)+(c_j-f_j(t)-p_j(t))/(W_j+1)]]. \quad (5.2.3)$$

The variable c_j in these equations represents the effective link capacity and not the true capacity. For the simulation we used $c_j=.8C_j$, where C_j is the actual capacity. These update functions differ slightly from those given earlier, in that we restrict the range of the controls. Since we know that the fair control allocation can never result in a control for link j outside the interval $[c_j/W_j,c_j]$ for Hayden's feasible set or $[c_j/(W_j+1),c_j]$ for Jaffe's feasible set, these are reasonable modifications.

A packet is an object, created by a user, that has a source and destination (two users), a route (a list of links), forward control and feedback information, and a variety of statistics, such as its length, time of generation, and the rate of its source.

When a packet is created, its forward control is set to infinity. Each time a packet is transmitted across a link, its forward control is reset to the minimum of its current forward control and the control of the link. Hence, when a packet arrives at its destination, its forward control is equal to the minimum control of the links in its path. This number is stored by the destination and used as the feedback value in the next packet created by the destination to be returned to the source. When the source receives the returned packet, it changes its rate to equal the feedback value.

Note that a packet contains its source's rate. This is desirable for two reasons. A link could calculate a given user's rate by multiplying the length of one of that user's packets by 50, but this produces serious round-off errors. Also, a user might sometimes want to transmit at a rate lower than that assigned, while still reserving for itself the option to send at the higher rate later. Hence, we prefer the users to communicate their rates to the links, rather than let the links measure the rates.

This completes the description of the simulation model.

5.2 The Simulation Program

In this section we describe the event driven program that was written to simulate the model of section 5.1.

The program works by scheduling and performing events in an event table. An EVENT consists of a TIME, a FUNCTION and ARGUMENTS. When an event is created, it is added to the *EVENT-TABLE*, which is implemented as a heap. The heap is sorted so that the event at the top of the heap is always the one whose TIME is earliest. An event "takes place" when it is removed from the top of the heap and its FUNCTION is applied to its ARGUMENTS. When this happens the global variable *TIME* is set to the TIME of the EVENT being performed.

The program begins by intializing the network, creating the start-up event and adding it to the heap. The program then enters a loop which repeatedly removes events from the top of the heap and performs them, until the global variable *TIME* exceeds the given time limit. Since most events, when performed, create one or more new events with times later than *TIME*, the heap never becomes empty.

There are ten different types of events that occur, as described below.

SIMULATION-STARTUP

This is the first event performed. It schedules the first VOICE-PACKET-GENERATIONs for all intially active users and the first CONTROL-PACKET-GENERATIONs for all initially silent users. While the first group of VOICE-PACKET-GENERATIONs are synchronized, subsequent VOICE-PACKET-GENERATIONs will rapidly fall out of synchronization. The same is true of the CONTROL-PACKET-GENERATIONs. This event also schedules the first UPDATEs and LINK-STATISTICS-COLLECTIONs for each link, and the first USER-STATISTICS-COLLECTIONs for each user. The times of the first link UPDATEs are randomized, since these events would not fall out of synchronization otherwise.

VOICE-PACKET-GENERATION

When a voice packet is generated, its FORWARD-CONTROL, FEEDBACK-CONTROL and LENGTH are set, as described in the previous section. The route of the packet is set to the route of its source, its GENERATION-TIME is set to *TIME*, and TYPE is set to VOICE. The LAST-IN-TALK-SPURT? flag is set to T or NIL, according to the outcome of a random "coin toss". The entire packet is scheduled to arrive at the first link in its route at *TIME* + *PACKET-GENERATION-DELAY*. This is accomplished by adding the events PACKET-ARRIVAL and PACKET-TAIL-ARRIVAL to the event-table. If LAST-IN-TALK-SPURT? is NIL, another VOICE-PACKET-GENERATION is scheduled for the appropriate future time, otherwise, a CONTROL-PACKET-GENERATION is scheduled.

CONTROL-PACKET-GENERATION

When this event is performed, the user first checks to see if it is active or not. If the user is active, then it started talking since the time at which the CONTROL-PACKET-GENERATION was scheduled, and no further action is performed. Otherwise, a control packet is generated, and its FORWARD-CONTROL, FEEDBACK-CONTROL and LENGTH are set, as described in the previous section. The route of the packet is set to the route of its source, its GENERATION-TIME is set to *TIME*, and TYPE is set to CONTROL. The packet is scheduled to arrive at the first link of its route at *TIME* + *PACKET-GENERATION-DELAY*, by adding the event PACKET-ARRIVAL to the event-table. Another CONTROL-PACKET-GENERATION is scheduled for the appropriate future time.

PACKET-ARRIVAL

When the head of a packet arrives at a link, its ARRIVAL-TIME is set to *TIME*, the packet is placed at the end of the queue, and link queue statistics are updated. If the link is idle when the packet arrives, a PACKET-TRANSMISSION is scheduled at *TIME* + *PACKET-TRANSMISSION-DELAY*.

PACKET-TAIL-ARRIVAL

When the tail of a voice packet arrives at a link, the link checks whether the packet's source is on its list of active users. If not, the link adds the source to its list. Then the link updates its stored value of the source's rate, which it reads from the packet. When the tail of a control packet arrives, PACKET-TAIL-ARRIVAL does nothing.

PACKET-TRANSMISSION

This event occurs either when the link transmits the tail of a packet, or when the head of a packet arrives at an empty queue. If the link is transmitting the tail of a packet when this event is performed, it will update its records of the number of bits transmitted. If the packet just transmitted was a voice packet and the last in its talk-spurt, the link will remove the packet's source from its list of active users.

Next, whether or not the link just finished a transmission, it checks its queue. If the queue is empty, the event is finished. Otherwise, the link gets the first packet in its queue and changes the packet's forward control value as described in the previous section. Then the link schedules the transmission of the packet's tail at

TIME + (PACKET-LENGTH/LINK-CAPACITY)
+ *PACKET-TRANSMISSION-DELAY. (5.3)

Next the link updates relevant statistics. Finally, the link removes itself from the head of the packet's route list, and checks for the packet's next destination. If the packet's route is empty, its next destination is its source's partner, where a PACKET-ABSORPTION is scheduled for

TIME + LINK-PROPAGATION-DELAY
+ *PACKET-ABSORPTION-DELAY*. (5.4)

Otherwise, the head of the packet is scheduled to arrive at the next link in its route at

TIME + LINK-PROPAGATION-DELAY
+ *PACKET-ARRIVAL-DELAY* (5.5)

and the tail of the packet is scheduled to arrive at

TIME + LINK-PROPAGATION-DELAY + *PACKET-ARRIVAL-DELAY*
+ (PACKET-LENGTH/LINK-CAPACITY). (5.6)

PACKET-ABSORPTION

This event represents the arrival of a packet at its destination. First, the packet's net delay is calculated by

DELAY=*TIME* + *PACKET-ABSORPTION-DELAY*
- PACKET-GENERATION-TIME. (5.7)

If it is a voice packet, the user's voice packet delay statistics are updated. Otherwise, the user's control packet delay statistics are updated. The user then sets its rate from the feedback information, and stores the forward control value for use as described earlier. If the packet is the last in its talk-spurt, a VOICE-PACKET-GENERATION is scheduled for the packet's destination.

UPDATE

When an UPDATE is performed for a link, the link first checks whether the update protocol is satisfied. If it is, the new CONTROL is calculated using the specified update function, and another UPDATE is scheduled for *TIME* + *UPDATE-INTERVAL*. Otherwise, another UPDATE is scheduled for *TIME* + *UPDATE-ATTEMPT-INTERVAL*.

LINK-STATISTICS-COLLECTION

This event collects link statistics, and adds them to the *LINK-STAT-STREAM*, which is output to a file. It then zeros the statistics and schedules the next LINK-STATISTICS-COLLECTION at *TIME* + *LINK-STATISTICS-INTERVAL*.

USER-STATISTICS-COLLECTION

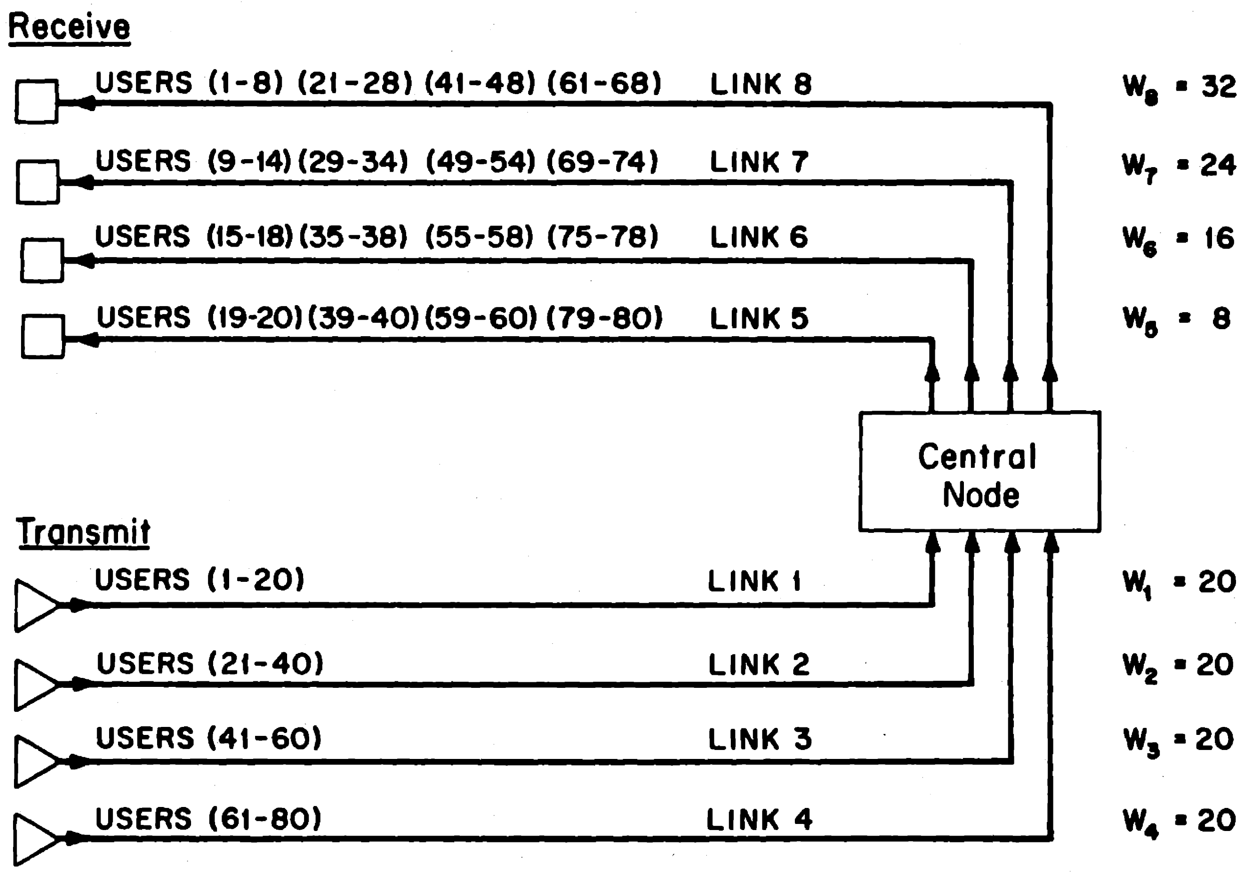
This event collects user statistics, and adds them to the *USER-STAT-STREAM*, which is output to a file. It then zeros the statistics and schedules the next USER-STATISTICS-COLLECTION at *TIME* + *USER-STATISTICS-INTERVAL*.

This completes the description of the simulation program.

5.3 The Network Model

In order to have a basis for comparison, we chose the same network model used by Hayden. This network is a scaled down version of a network simulated at Lincoln Laboratories [16]. The network consists of 80 users and 8 links, and its topology is illustrated in Figure 5.0. The original network model considered traffic flow in only one direction, but in order to model the effects of feedback delay we must consider two-way traffic flow. In order to use this model for two-way traffic without incorporating additional links, we view all the sources as being at the same location.

We make the same user-partner assignment as Hayden, where each user i has as its partner user (81-i), for i=1,...,40. Ideally, there should be no correlation between the set of links in a user's path and the set of links in its partner's path, as would be the case if we had incorporated extra links to handle the two-way traffic. However, the user pairs



The Simulation Network

Figure 5.0

(32,49), (31,50), (30,51) and (29,52) all share link 7. But, since they constitute only 8 out of 24 users, this should not be a serious problem.

The program as described in the previous section contains many variables related to the actual performance of a physical network. The values of these variables are given in Table 5.1, and have been chosen to be consistent with current technology.

5.4 Simulation Results

In this section we describe the results of the simulation. The theory of the preceding chapters addresses only the the behavior of networks with fixed configuration. In practice, however, the network configuration will change rapidly as users initiate and end conversations. The ability of an algorithm to control the link flows in a changing network depends upon the rate of convergence of flows in a static network. If the time required for the controls to converge in a static network is short compared to the rate of change of the dynamic network, the algorithm will work for the dynamic network. Hence, we divide our results into two subsections: static results and dynamic results.

5.4.1 Static Results

Variable	Value
LINK-CAPACITY	40000 bits/sec
LINK-PROPAGATION-DELAY	.003 sec
PACKET-ABSORPTION-DELAY	.0005 sec
PACKET-ARRIVAL-DELAY	.0005 sec
PACKET-TRANSMISSION-DELAY	.0001 sec
PACKET-GENERATION-DELAY	.0005 sec

Network Constants

Table 5.2

In all of the experiments described in this section, we studied a static network in which all even numbered users were active for the entire simulation. For this network, Hayden's fair control vector is (8000,8000,8000,8000,8000,4000,2667,2000) and Jaffe's fair control vector is (4987,4987,4987,4987,12050,3555,2461,1882).

Two important parameters to adjust are the update interval and the update attempt interval. Preliminary results indicated that the update attempt interval should be kept as small as possible. While we could set the update attempt interval so that each link tries to update each time it receives a packet, this would slow down the simulation considerably. Instead, we set the update attempt interval to 20 msec., so that each link tries to update after receiving a new packet from each of its users.

Hayden observed, in his simulations, that setting the update interval to 20 msec. produced severe oscillations in the link flows. With an update interval of 100 msec., these oscillations were greatly reduced. For comparison, we ran our simulation using each of these values.

We have identified three parameters to adjust. We may choose between the HAYDEN, JAFFE or MOSELY update functions, the update protocol or no update protocol, and fast (every 20 msec.) or slow (every 100 msec.) updates. We ran the simulation for each of the twelve combinations.

For the sake of brevity, we have chosen to display the results for links 2 and 5 only: link 2 because it is typical of the others, and link 5 because it is atypical. Link 5 differs from the rest in that, when the rest of the links' controls correspond to Hayden's fair allocation, link 5 controls none of its users. Thus Hayden's algorithm will assign link 5 a control equal to its capacity..

Figures 5.1 - 5.3 show link 2 controls versus time for all possible combinations of update function, update protocol and rate of update. Figures 5.4 - 5.6 show the same data for link 5. Inspecting these figures, we make the following observations.

Both the HAYDEN and JAFFE update functions work moderately well with slow updates on link 2. The JAFFE update function can also control link 5 with slow updates, but the HAYDEN function cannot. Both functions seem to work marginally better with update protocols than without. Neither function works at all well with fast updates, with or without the protocol, although the protocol tends to damp the oscillations for the JAFFE function. These results are largely what we expected, based on Hayden's simulations. It is a little surprising, however, in view of the theory of Chapter 3, that the JAFFE function performs so poorly with fast updates and the update protocol. These results do not contradict the theory, though, since the theory makes no claims about the rate of convergence, and the controls do converge.

The MOSELY update function is capable of controlling both links well under all circumstances. Convergence of the controls with slow updates is slightly faster without the update protocol than with. Since the update protocol is not necessary for the controls to converge, it is not surprising that the protocol slows convergence down, as the protocol must occasionally prevent a possibly beneficial update.

When updates are fast, there is almost no noticeable difference between the performance of the algorithm with or without the protocol. For link 2, it might be argued that, because the controls converge to the fair controls from below, the update protocol is nearly always satisfied, and hence the performance is the same with or without the protocol. In order to test this theory, we ran the simulation with the initial controls and rates set high (4000 bits/sec.) to try to produce a control sequence that converged from above. The results of that simulation are shown in Figures 5.7 - 5.8. The controls still converge from below, since after starting high, the link cuts its control sharply to limit the flow.

There is more difference between the performance with and without the protocol in this case, but the difference is still small.

From these figures we conclude that the HAYDEN and JAFFE update functions work best with slow updates, and the MOSELY function works best with fast updates. In Figures 5.9 - 5.12, we compare the performances of the three functions. Since the JAFFE update function is designed to converge to a different fair allocation than the others, it is difficult to assess how well it does this relative to the other two functions. From Figures 5.9 and 5.10, we might conclude that the MOSELY function performs no better than the HAYDEN or JAFFE functions. But Figures 5.11 and 5.12 show that this is not so. Collectively, these figures seem to indicate that under best case conditions, the functions work approximately equally well, but for unusual conditions, the MOSELY function works better.

As further evidence of this conclusion, consider Figures 5.13 and 5.14. These figures show the link 2 controls produced by the HAYDEN and MOSELY functions, for four different sample simulation runs for each. The HAYDEN update function gives varying results for each run, while the different control sequences produced by the MOSELY function are indistinguishable. The variability of the HAYDEN function is more surprising than the consistency of the MOSELY function, when we consider how little difference there is in the loads offered to the network for each sample run. For the static simulations, the pattern of conversations does not change, only the order of arrival of the packets in the queues and the order of link updates differ from one run to the next.

The static simulation results seem to indicate that the MOSELY update function is slightly superior to the others. However, the real test is how well the functions perform for a dynamic network.

5.4.2 Dynamic Results

In this section we describe the results of the dynamic simulation experiments. We preface this discussion by remarking that the simulation results are sufficiently unexpected that we suspect an error in the program, and we feel that additionaly testing is called for.

For all the simulation runs described in this section, we used an average talk-spurt length of 60 packets, corresponding to a 1.2 second talk-spurt duration. Thus, the link loads change rapidly, as one user stops talking and its partner begins. We ran the simulation for six different algorithms, using each of the three update functions with and without the update protocol. The MOSELY function was simulated only for fast updates, and the HAYDEN and JAFFE functions only for slow updates, since we found from our static experiments that that is how these functions perform best. In each case, we simulated the network for a 30 second interval.

For the sake of brevity, we show results only for links 2 and 8. By the symmetry of the network, links 1 - 4 are all essentially equivalent, and we choose link 2 as representative of the others. Link 8 is the most heavily loaded link, and we choose it for worst case behavior. Figures 5.15 - 5.26 show, for these two links and six algorithms, the number of bits transmitted over a 0.10 second interval, along with the number of active users and the maximum queue size, as a function of time. So that these three quantities may all be displayed on the same graph, we have scaled the number of users up by a factor of 100, and the maximum queue size by a factor of 10.

In Figure 5.15 we notice that for times between 20 and 25 seconds, the flow appears to exceed the link's capacity! It only appears this way, however, because of the way that the links count the bits transmitted. After each packet is transmitted, the link increments its bit count by the length of the packet. This count is zeroed at the end of each 0.10 second interval. Thus a packet which begins transmission in one statistics collection interval, and finishes in another, will count as having been wholly tranmitted in the second interval. This is why we see a sequence of intervals where the flow fluctuates above and below link capacity.

From examining these figures we see that the HAYDEN and MOSELY update functions produce totally unacceptable queues. The queues for the JAFFE function are much smaller. These results are unexpected and we cannot explain them. We find the size of the queues somewhat surprising, since Hayden's simulations rarely showed queues exceeding 50 packets. One possible explanation for this difference is that Hayden never ran his simulation for more than a 10 second interval. We notice that the really severe queues don't generally occur till after 10 seconds. This could be due to the fact that the simulation starts with all rates and controls relatively low, and only the even numbered users active. Hence, it may take several seconds for these effects to die out and steady state behavior to dominate. Also, Hayden used measured flows to calculate the controls while we calculate controls using a theoretical flow based on the sum of the users' rates. Since these calculated flows do not reflect the presence of control packets in the network, the resulting controls will be somewhat conservative. This seems like it should be a second order effect, however.

In general, the queues build up in response to sudden large changes in the number of active users. We see from the static results that it takes all of the algorithms around 2 - 3 seconds to converge to the correct flow, so, when the number of users changes rapidly, the algorithm cannot cope with the change. We also observe that the queues are much worse for the algorithm with protocol than without. This is not surprising, since the protocol prevents a link from lowering its control until all its users are sending at a rate lower than its control.

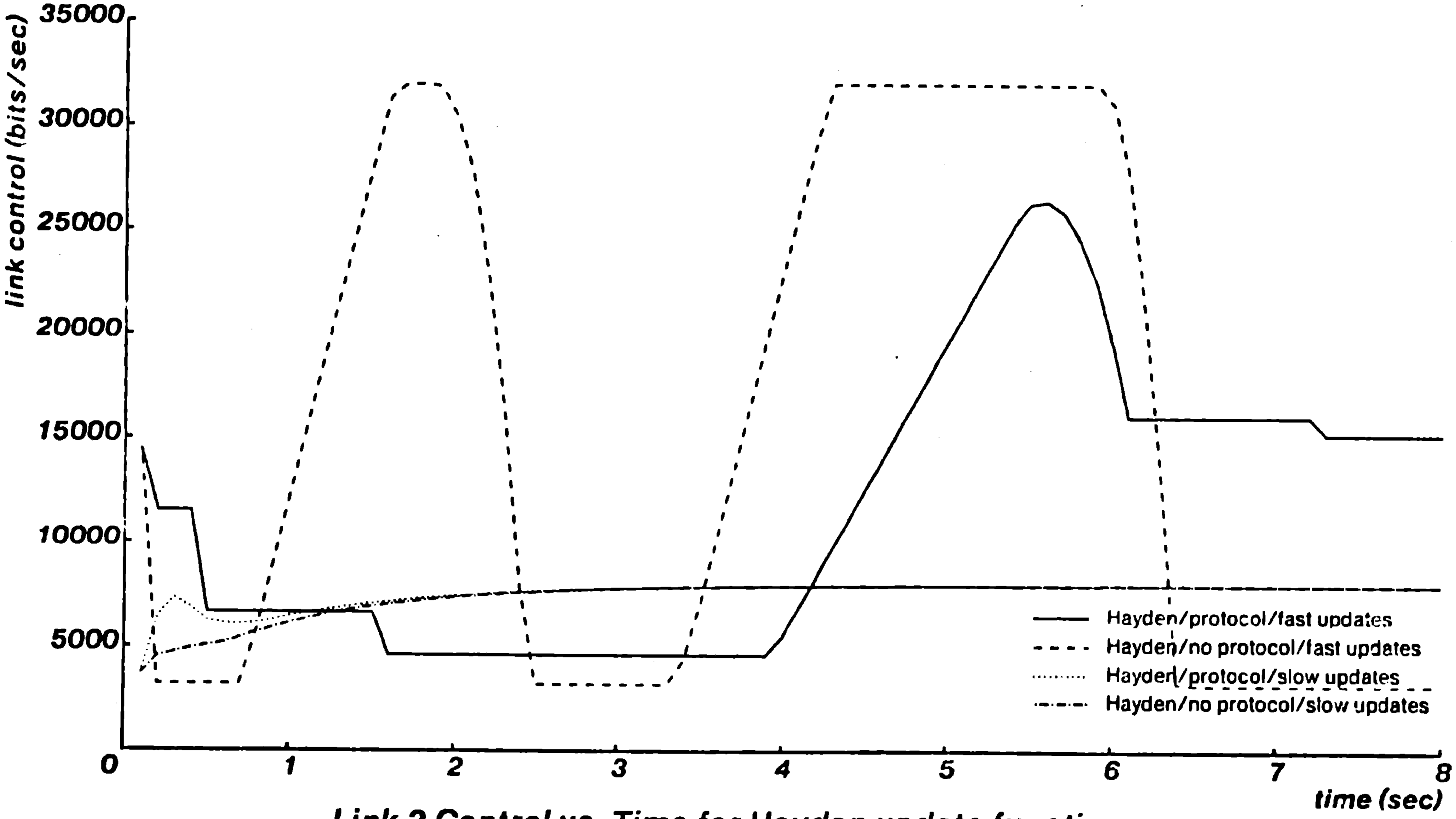
It is interesting to note that, for the JAFFE function, the flows seems to oscillate more in response to the changes in the number of active users. These oscillations are similar to the oscillations observed for the static simulations.

Observing that the JAFFE function produced more stable queues than the other functions, and that the MOSELY function tended to result in higher flows, we ran the simulation for a seventh, hybrid algorithm. The update function for the algorithms is obtained by replacing the value of the link control wherever it appears in the JAFFE function with the value of the maximum user rate. The hybrid algorithm was implemented without protocol and with fast updates. The results are shown in Figures 5.27 and 5.28. Unfortunately, there is no distinct improvement in performance for this algorithm.

One possible explanation for the difference in queue sizes for these algorithms, is that the fair flows for the JAFFE function are smaller than the fair flows for the other functions. Hence, we expect smaller queues. To examine this effect, in Figures 5.27 and 5.28, we plot average delay versus average flow for each of the seven algorithms. The averages are computed over the entire 30 second interval. While the average flows are indeed smaller for the JAFFE and HYBRID functions, they are only slightly smaller. The difference in average delays, however, is very great, and we are forced to conclude that the JAFFE function gives inherently better delay performance.

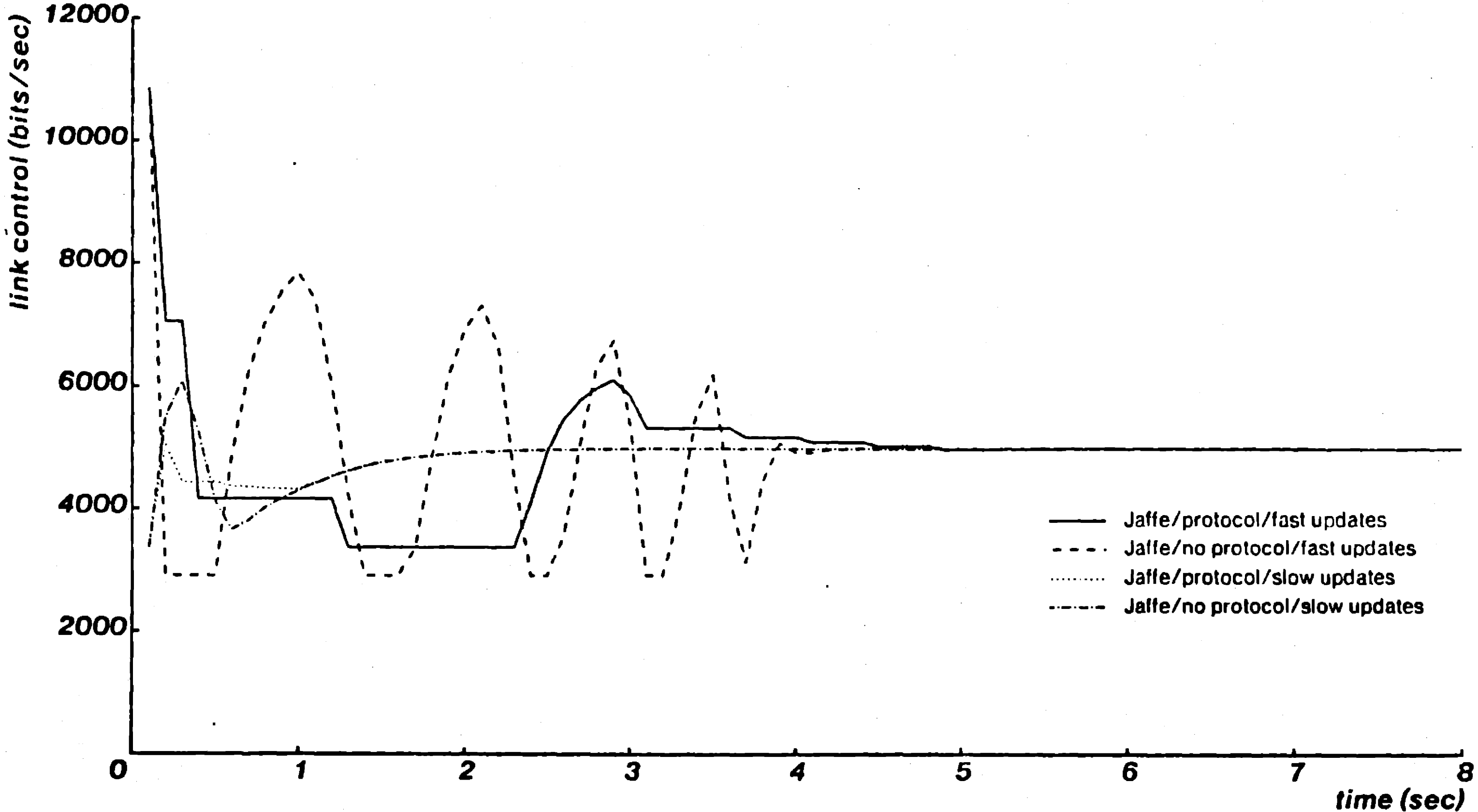
We believe that the intrinsic problem with all the algorithms thus far proposed is that convergence of the controls under static network conditions is too slow when compared with the rate of change we may expect in a dynamic network. One reason for this is the long feedback delay between the links and the sessions. Two things could be done to remedy this. Control packets could be generated more often, though this would increase overhead, and control packets could be given priority in the queues.

Another reason that convergence is slow is that the changes that a link makes in its controls are always conservative. Essentially, the links assume that all of their users will be affected equally by changes in controls. But this assumption is clearly erroneous. For example, if a link decides to lower its control from p_1 to p_2, none of its users that are currently sending at rates lower than p_2 will be affected. Similarly, if a link raises its control when most of its users are sending at much less than the current control, it is unlikely that any of those users will be affected by the change. With the HAYDEN update function, the links ignore potential feedback information from the users by taking note of only the sum of the rates. The MOSELY update function makes only slightly more use of the available information, by observing the sum of the rates and the maximum rates. Perhaps a really effective update function could be devised, where the links make use of the entire rate vector.



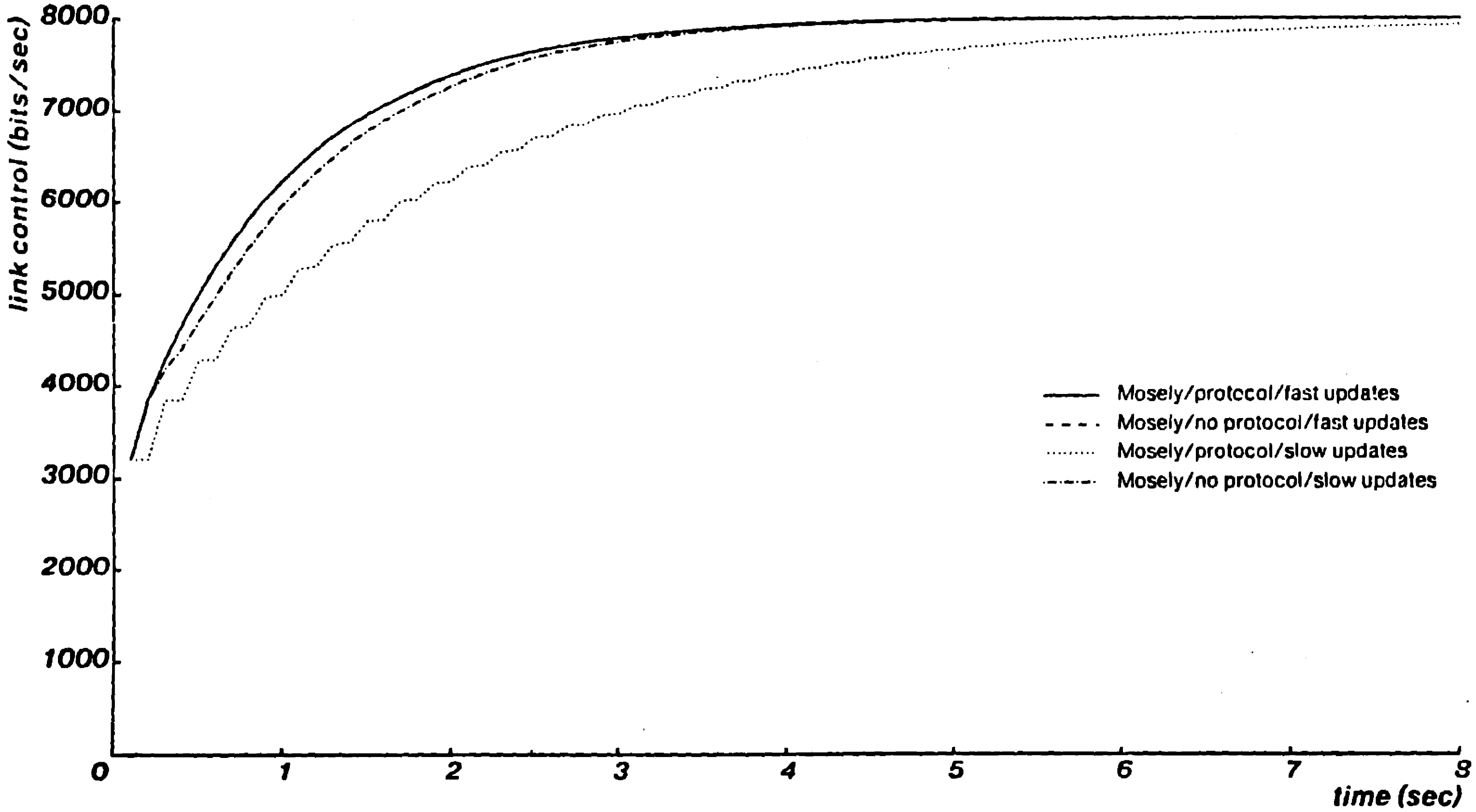
Link 2 Control vs. Time for Hayden update function (static network)

Figure 5.1



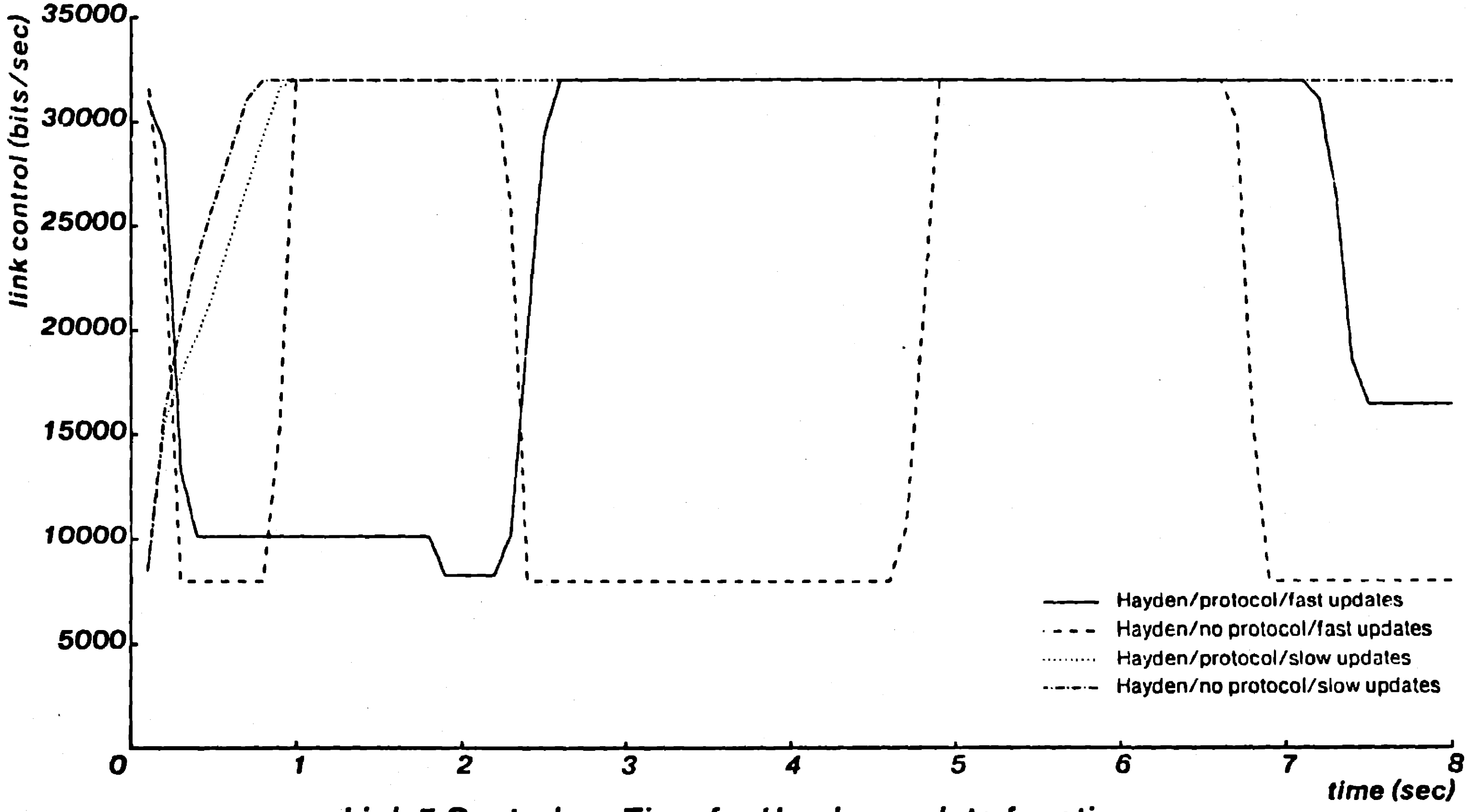
Link 2 Control vs. Time for Jaffe update function (static network)

Figure 5.2



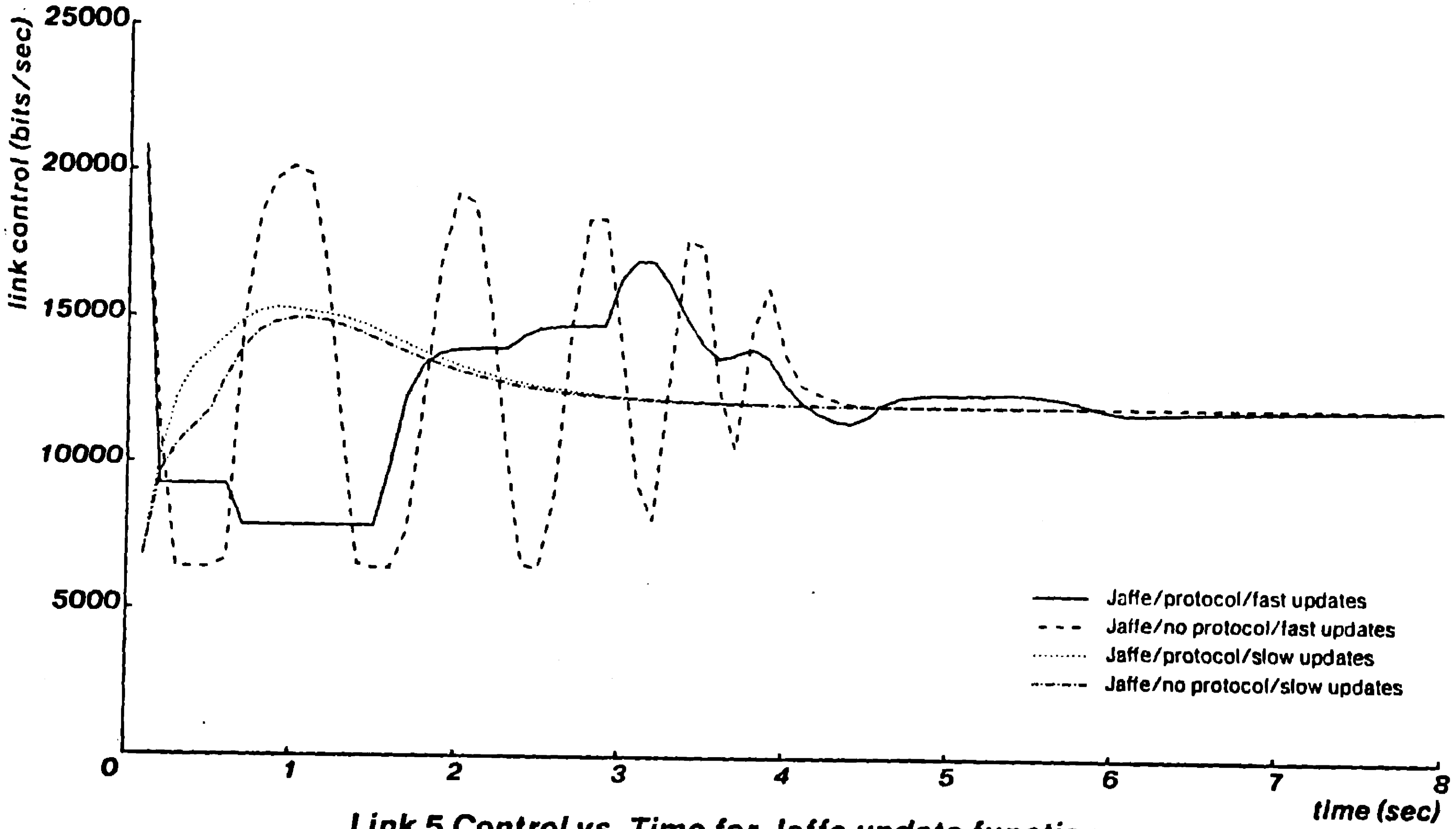
Link 2 Control vs. Time for Mosely update function (static network)

Figure 5.3



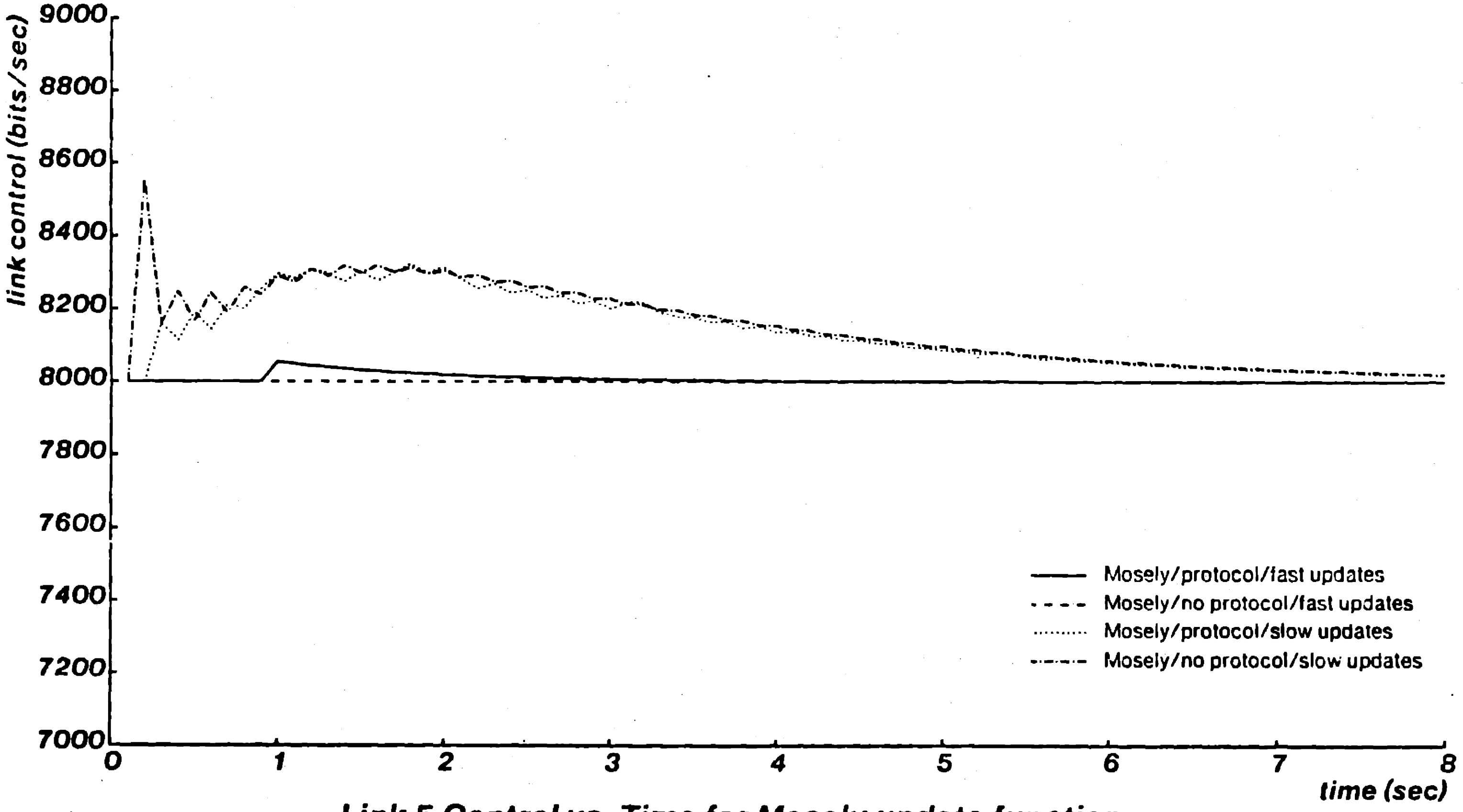
Link 5 Control vs. Time for Hayden update function
(static network)

Figure 5.4



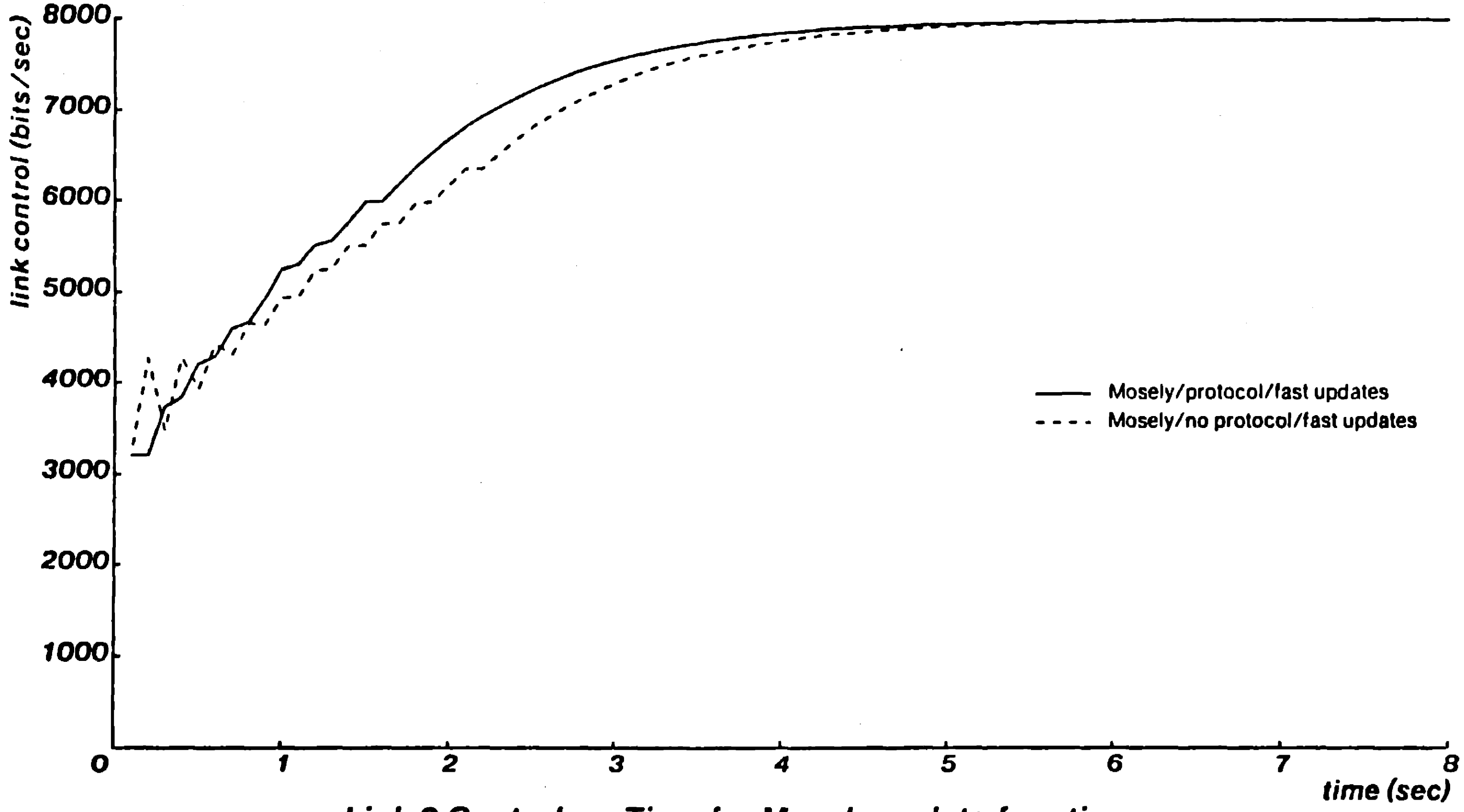
Link 5 Control vs. Time for Jaffe update function (static network)

Figure 5.5



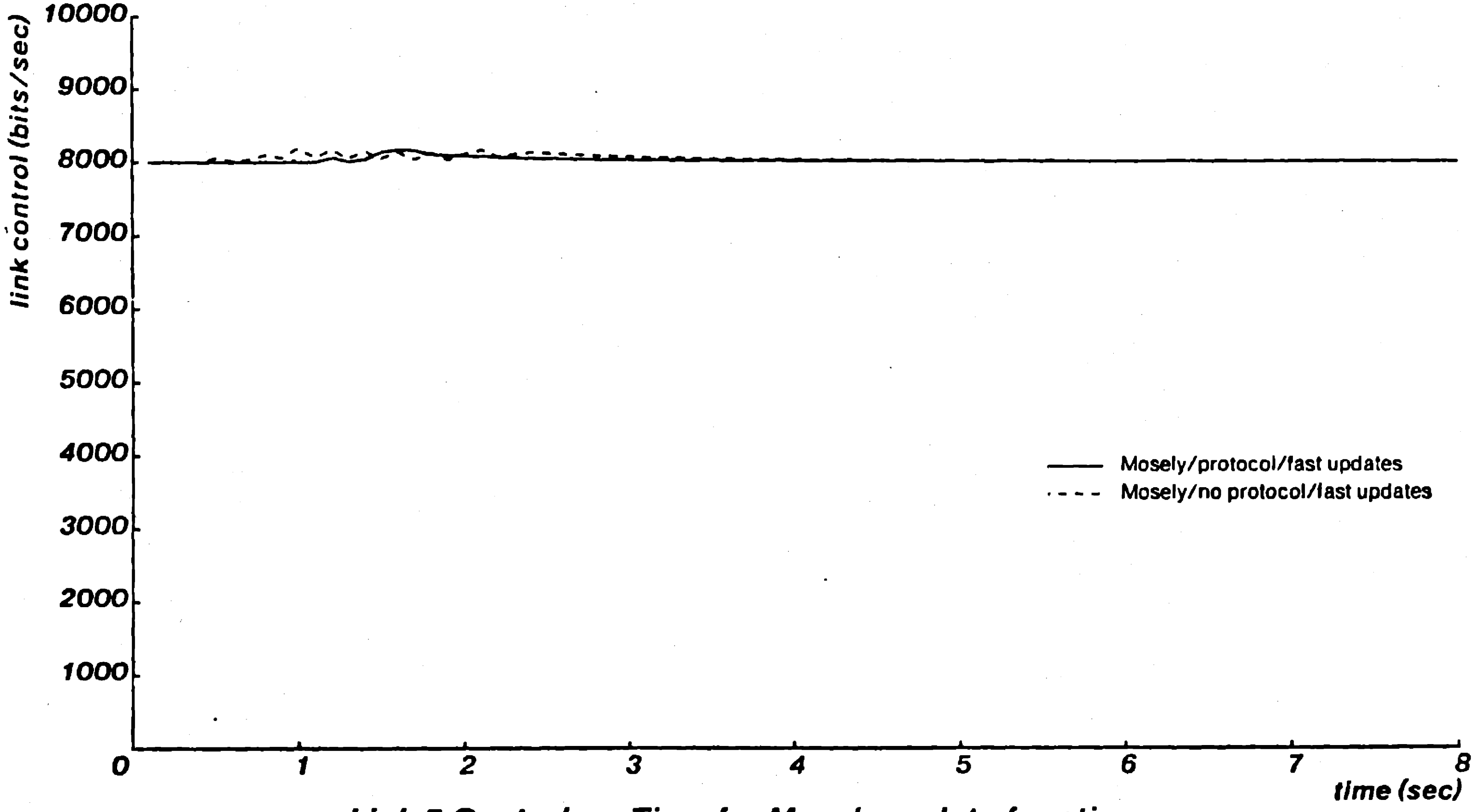
Link 5 Control vs. Time for Mosely update function (static network)

Figure 5.6



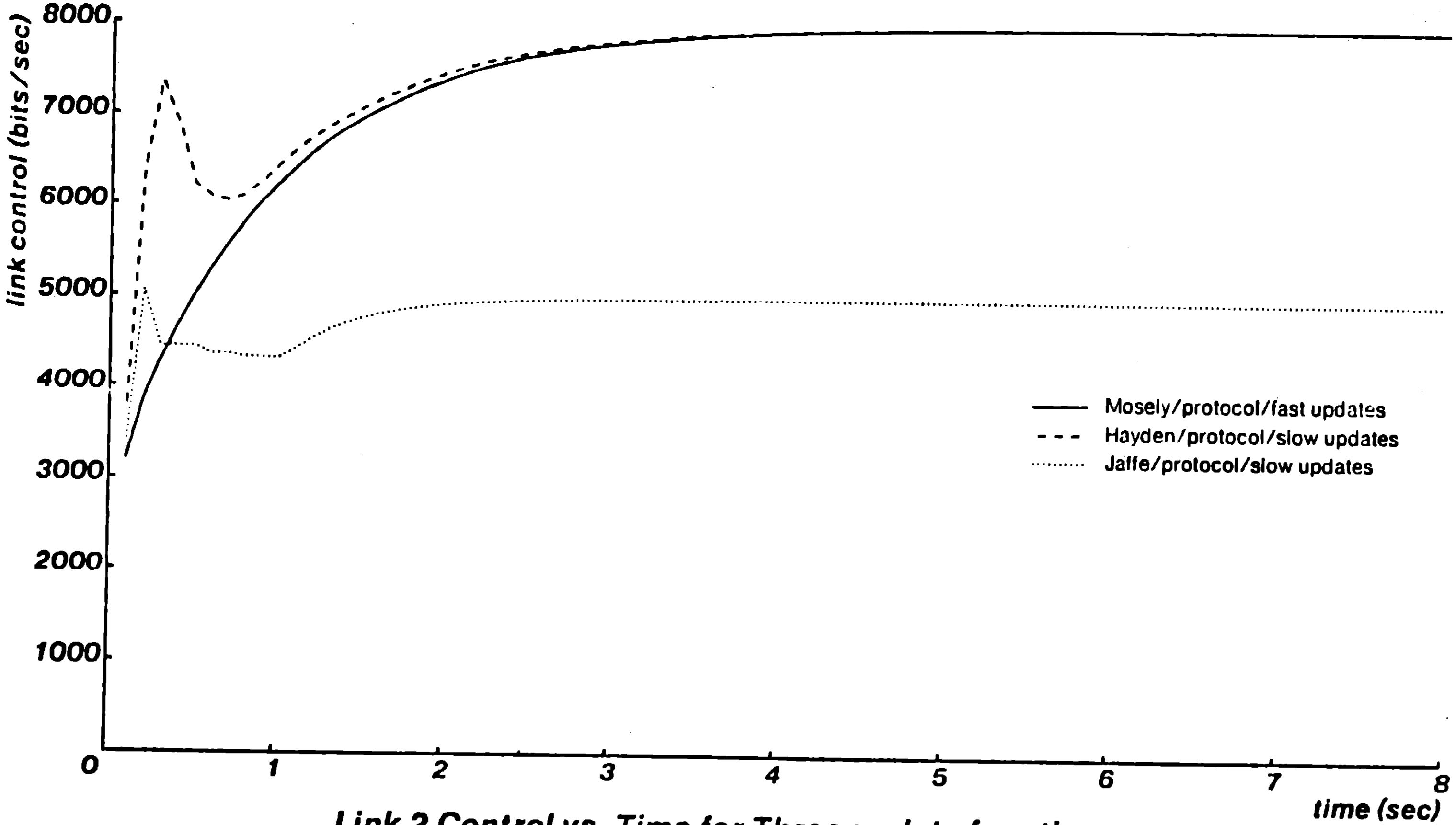
Link 2 Control vs. Time for Mosely update function
(static network)

Figure 5.7



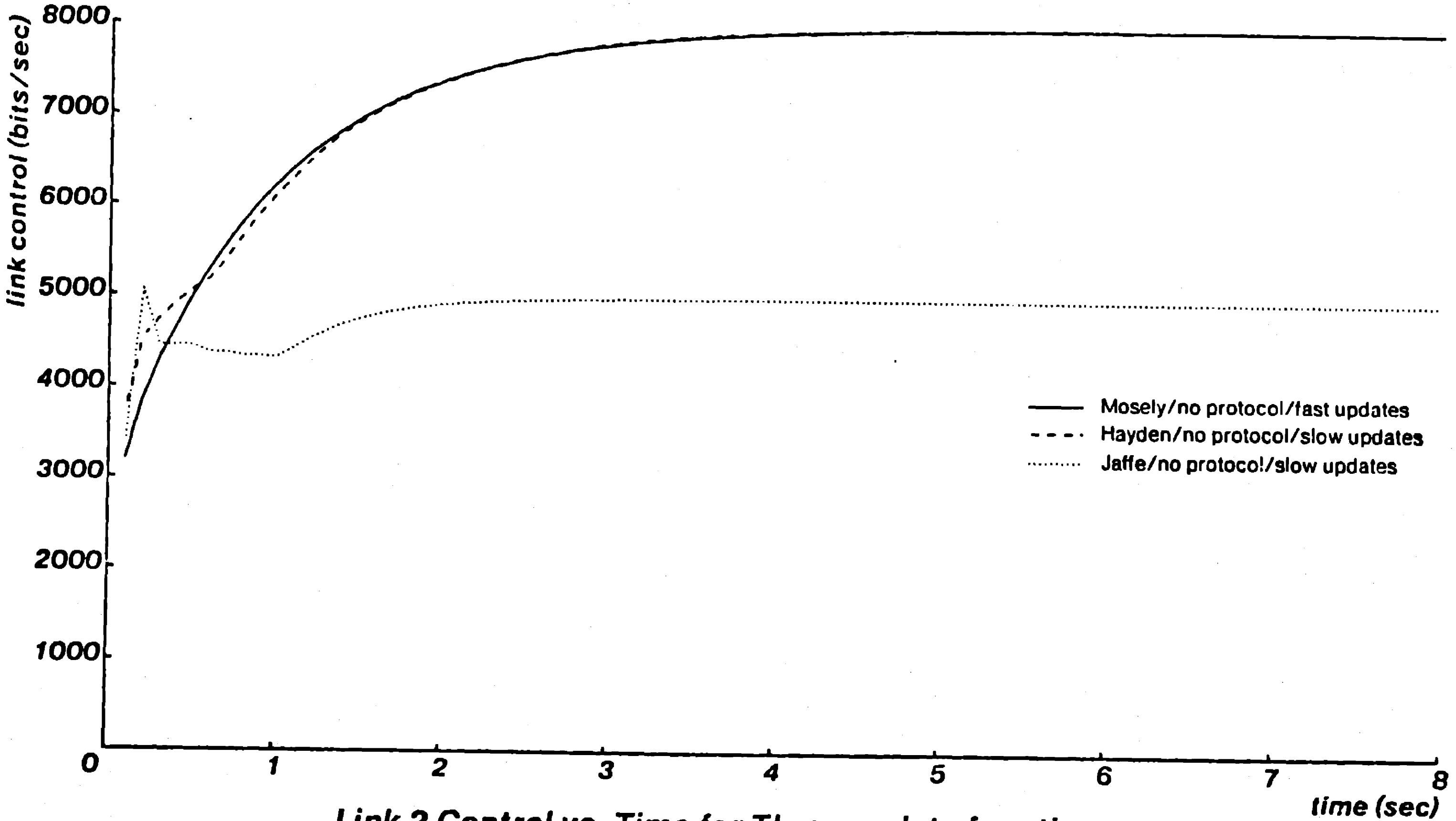
Link 5 Control vs. Time for Mosely update function (static network)

Figure 5.8



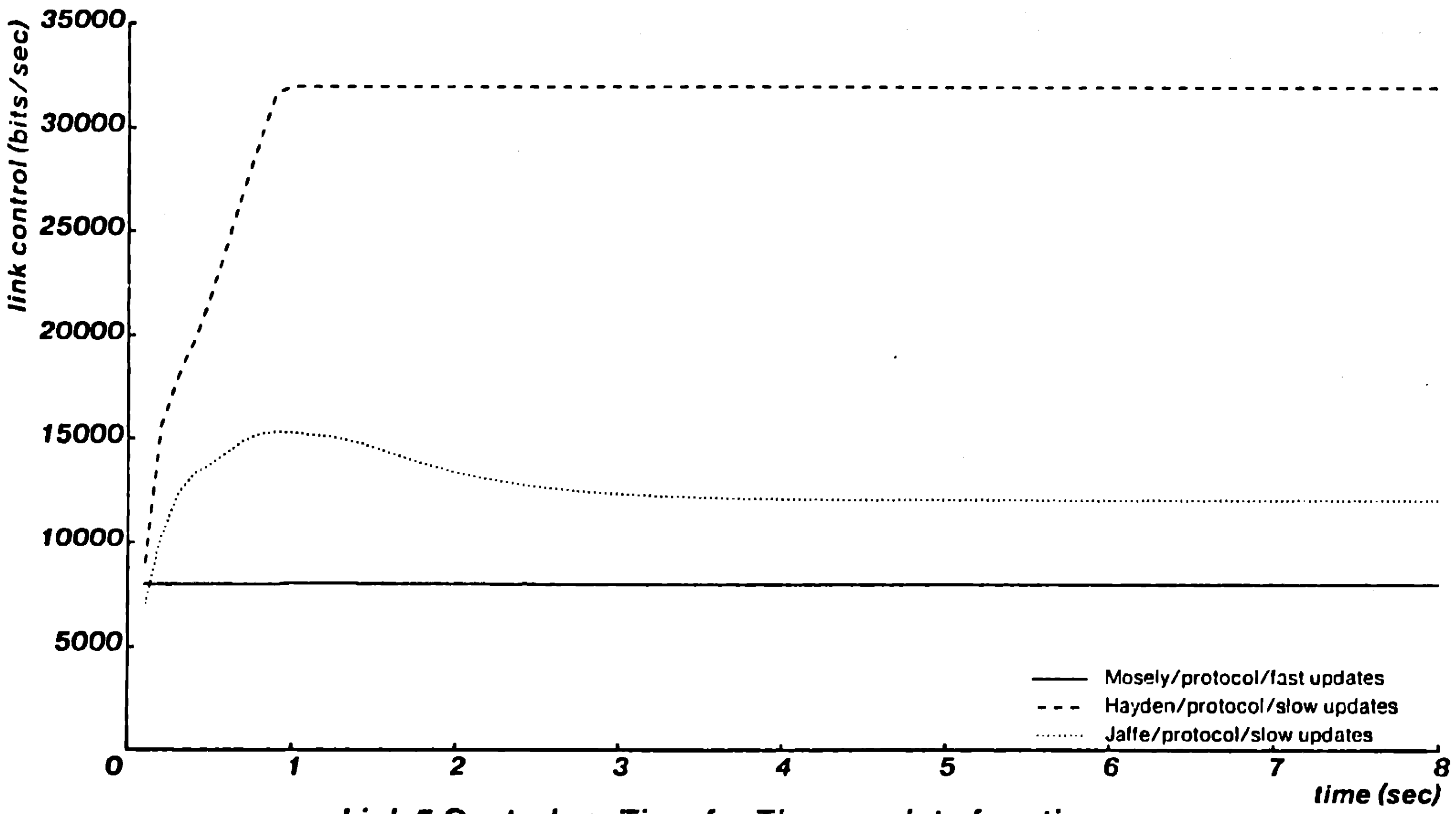
Link 2 Control vs. Time for Three update functions (static network)

Figure 5.9



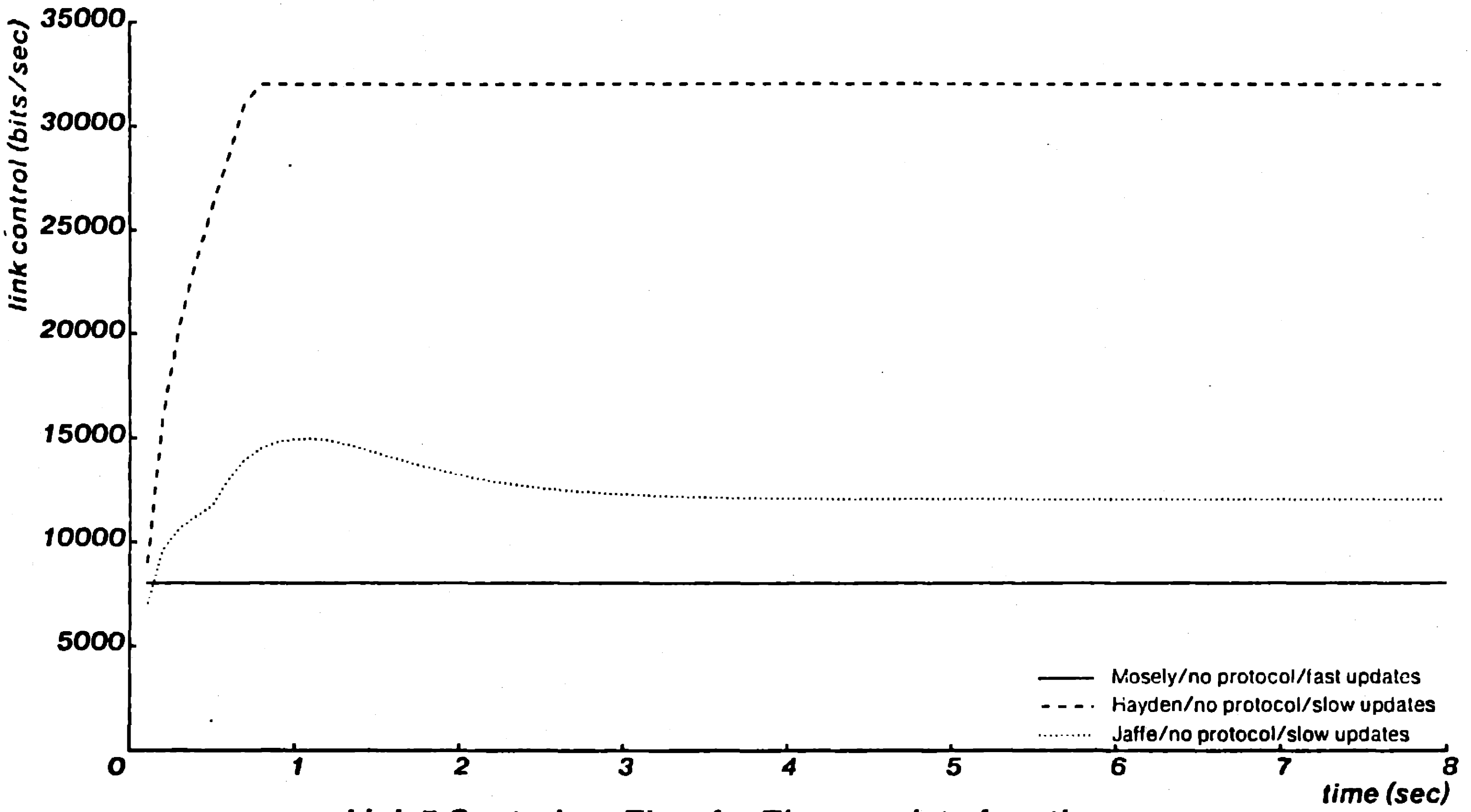
Link 2 Control vs. Time for Three update functions (static network)

Figure 5.10



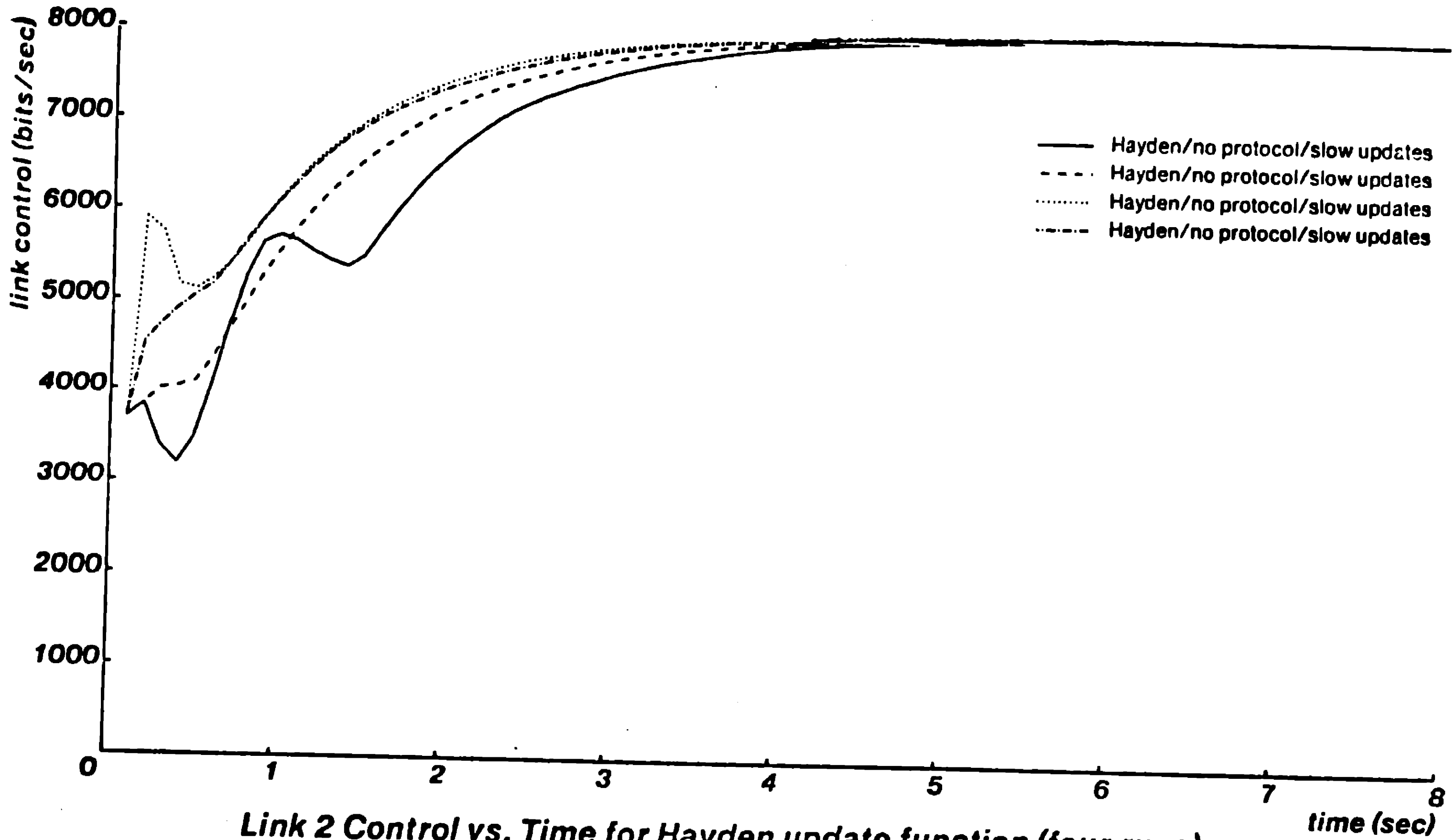
Link 5 Control vs. Time for Three update functions
(static network)

Figure 5.11



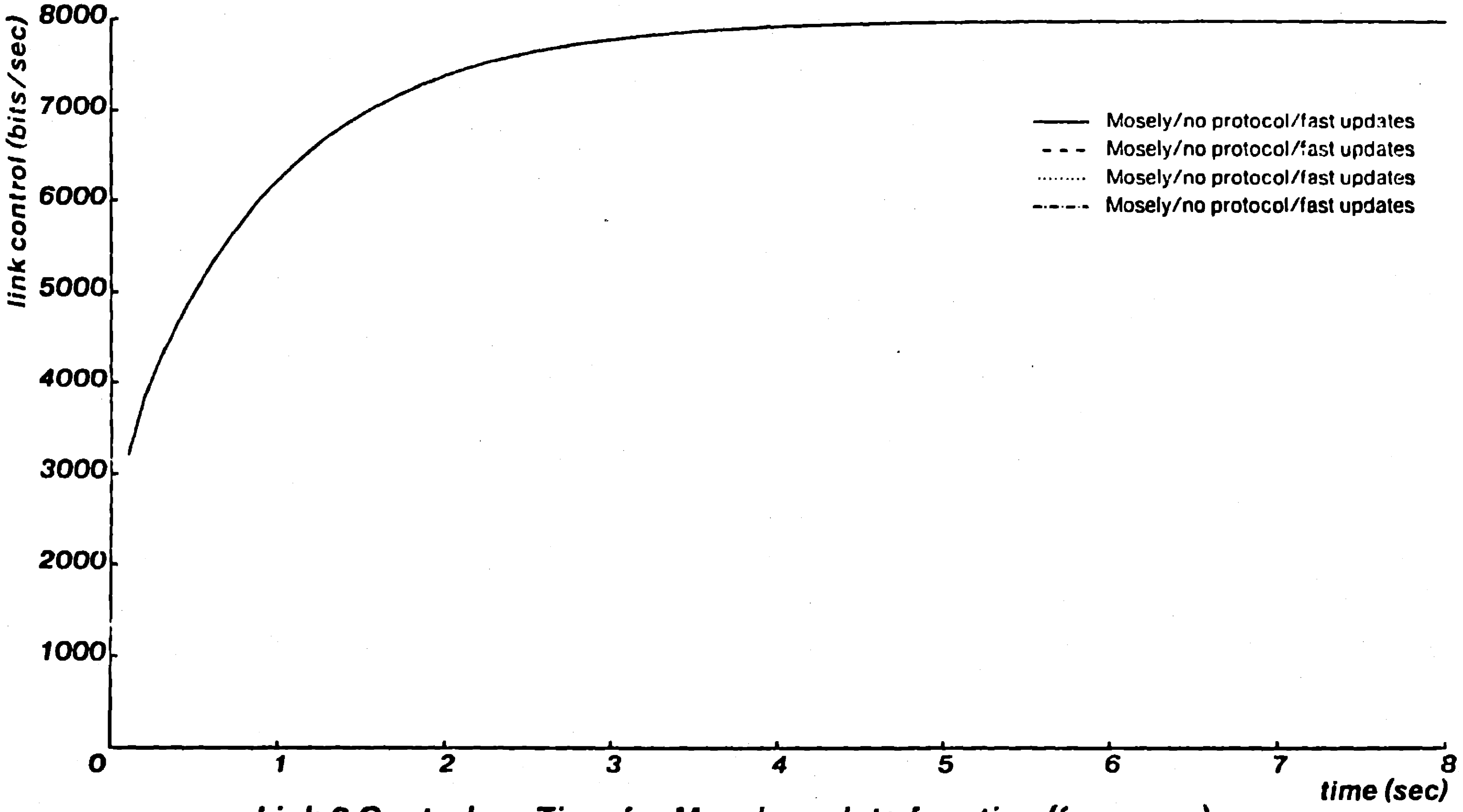
Link 5 Control vs. Time for Three update functions (static network)

Figure 5.12



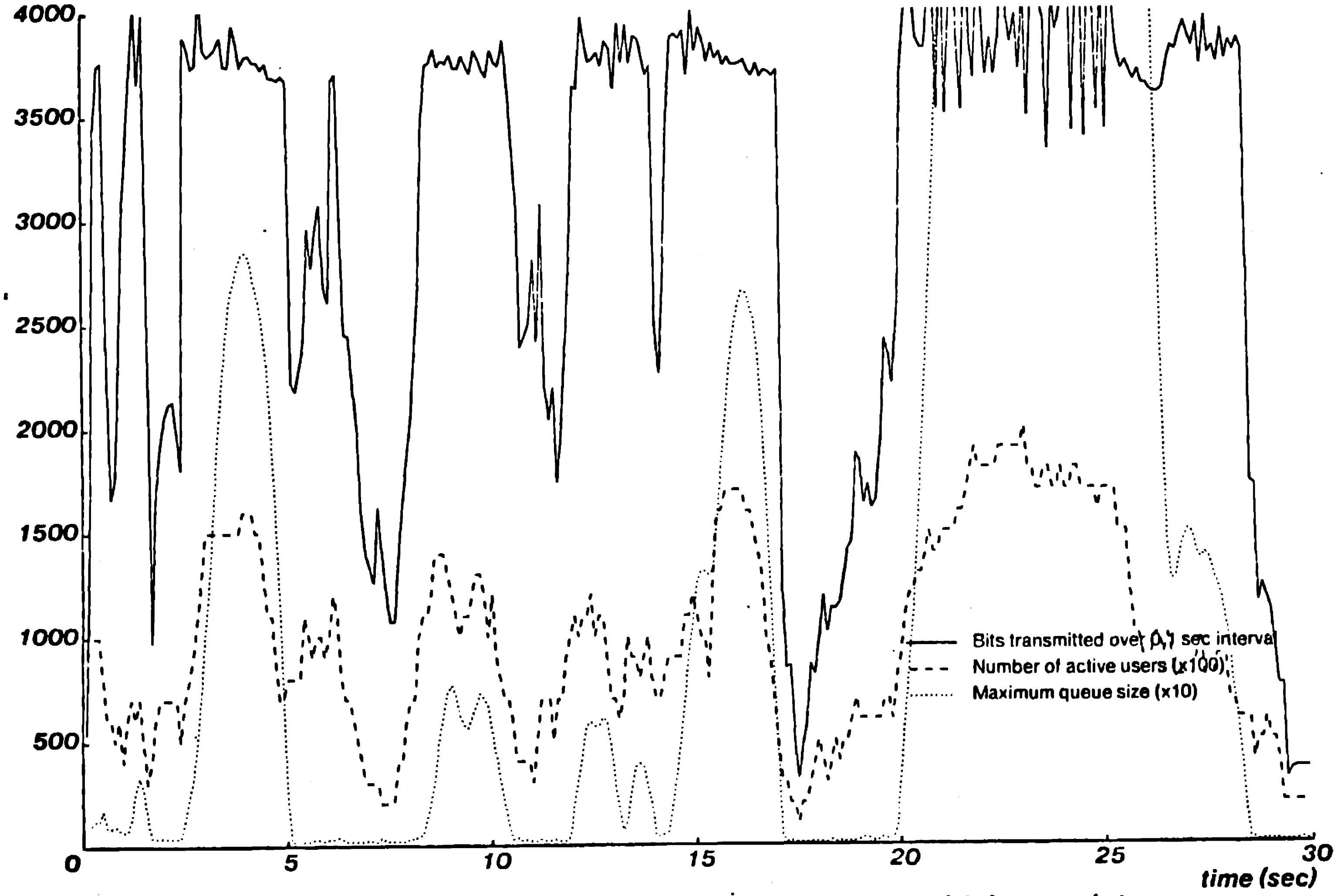
Link 2 Control vs. Time for Hayden update function (four runs)
(static network)

Figure 5.13



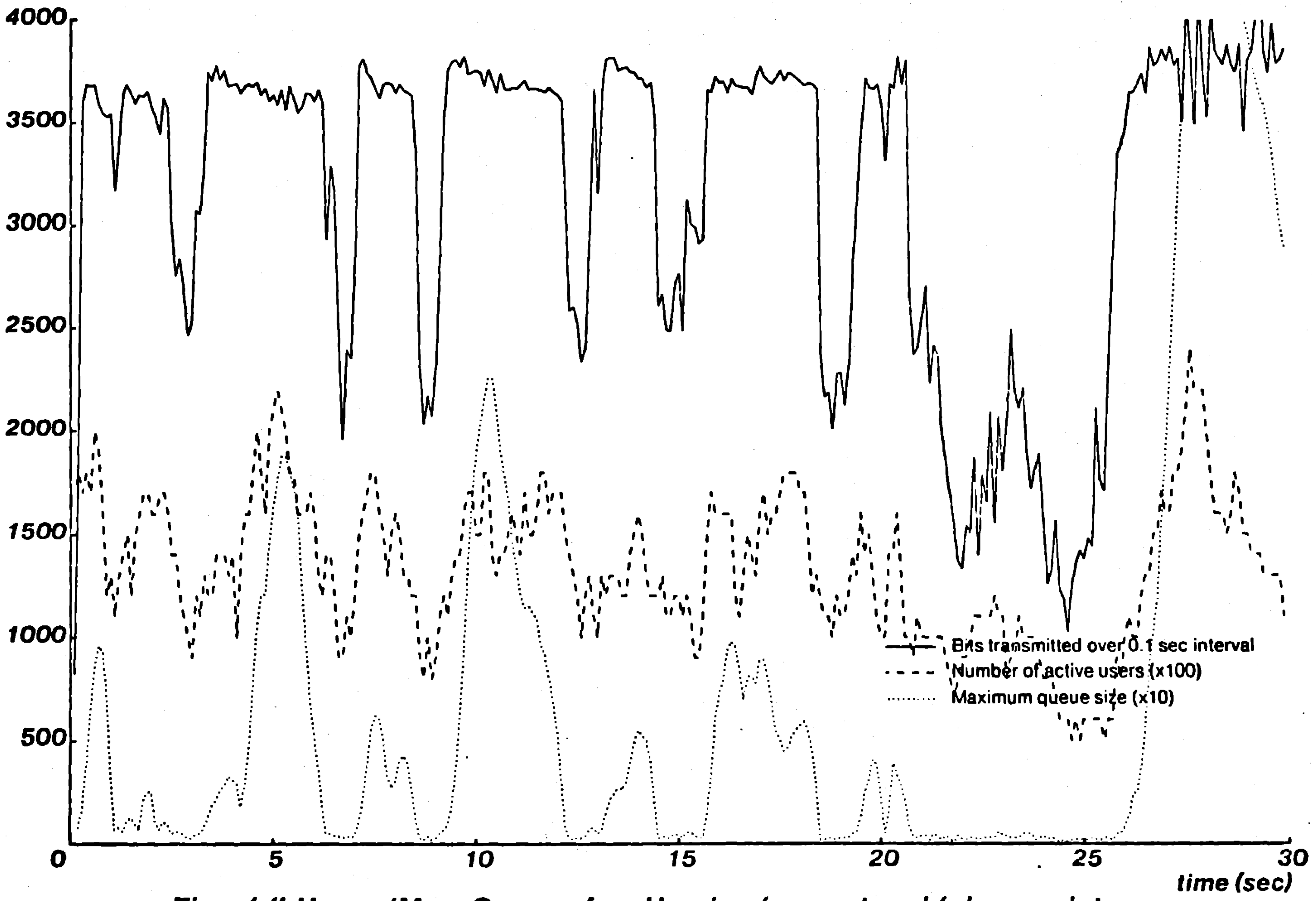
Link 2 Control vs. Time for Mosely update function (four runs)
(static network)

Figure 5.14



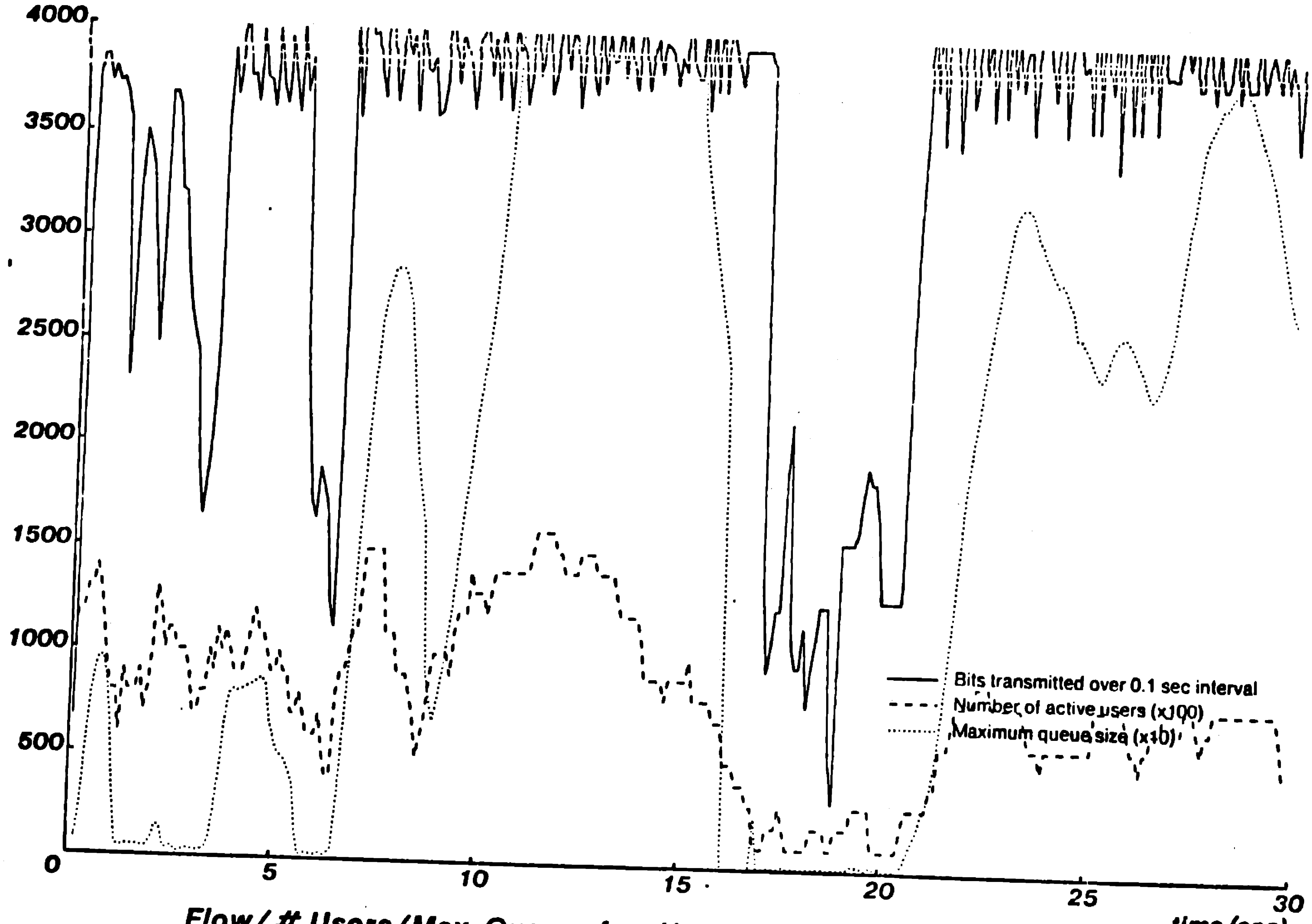
**Flow/# Users/Max. Queue for Hayden/no protocol/slow updates
Link 2 (dynamic network)**

Figure 5.15



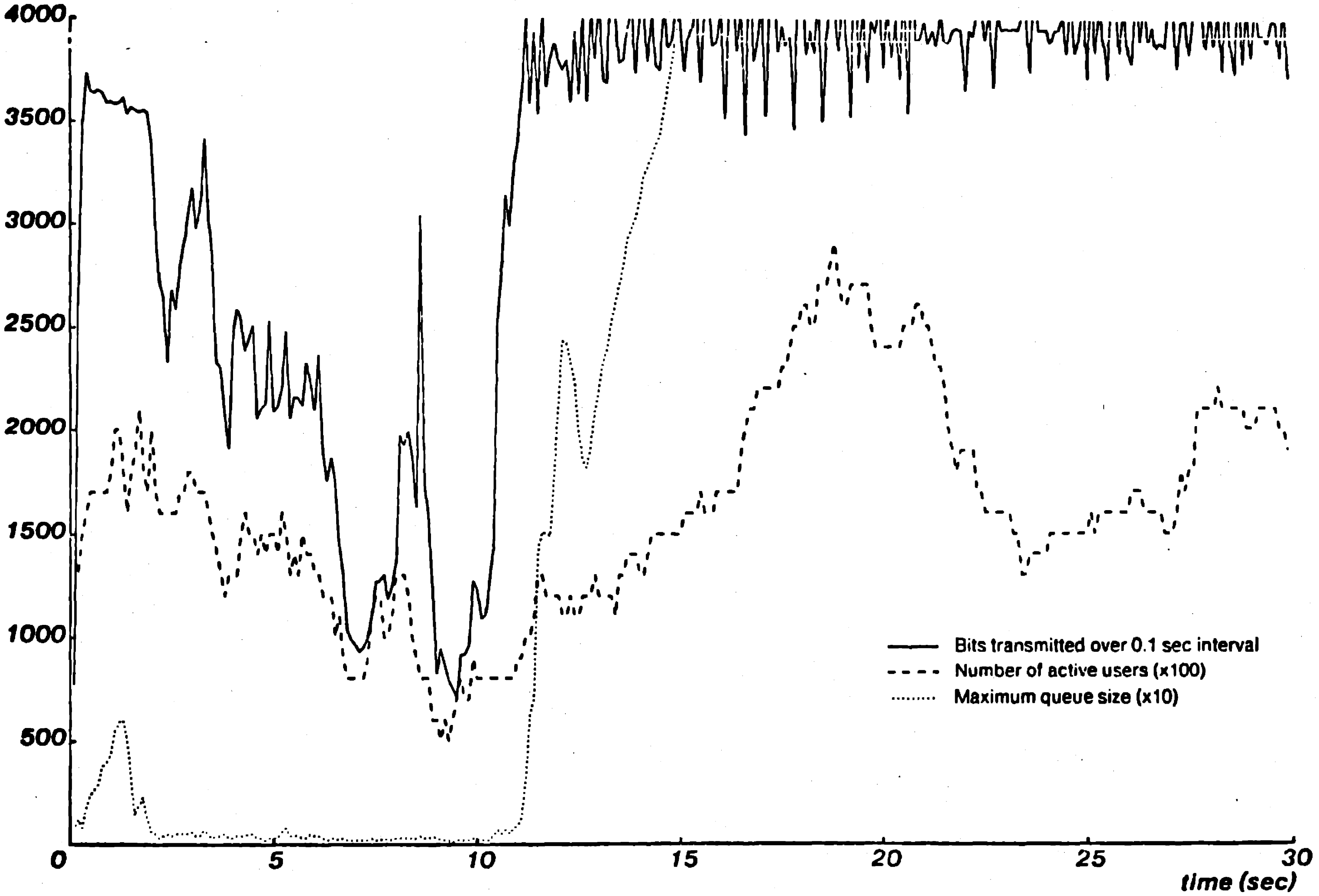
**Flow/ # Users/Max. Queue for Hayden/no protocol/slow updates
Link 8 (dynamic network)**

Figure 5.16



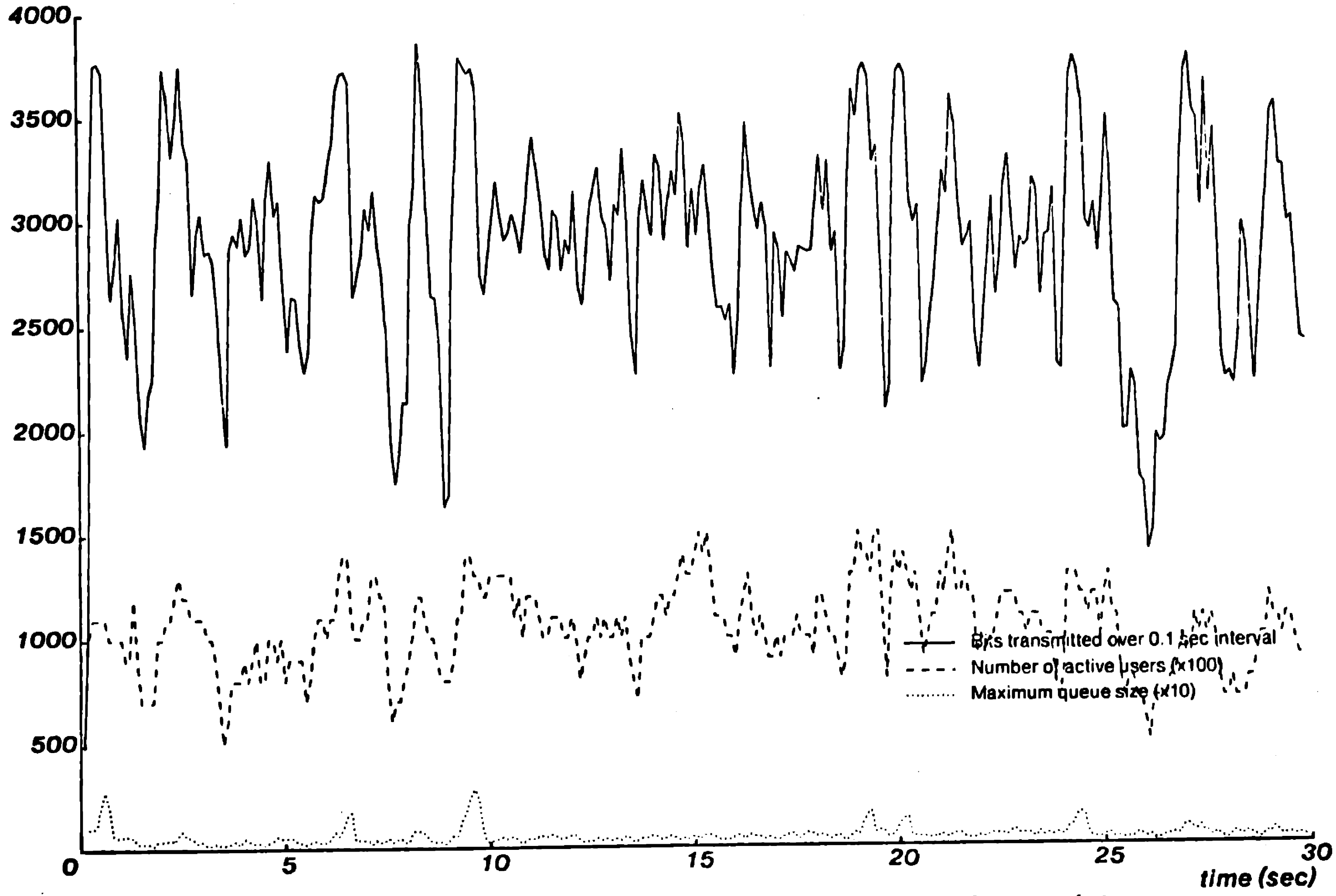
Flow/# Users/Max. Queue for Hayden/protocol/slow updates
Link 2 (dynamic network)

Figure 5.17



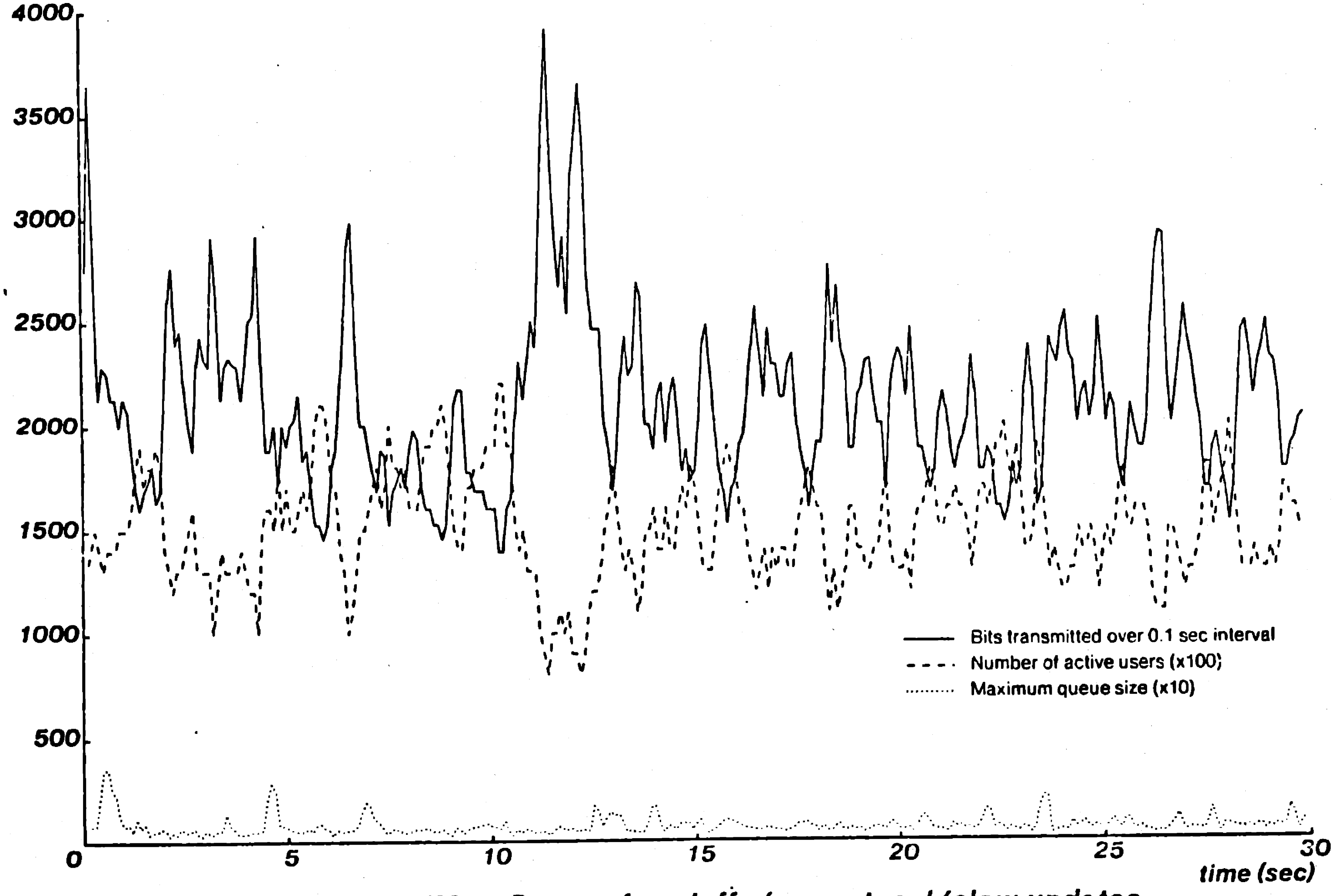
Flow/# Users/Max. Queue for Hayden/protocol/slow updates
Link 8 (dynamic network)

Figure 5.18



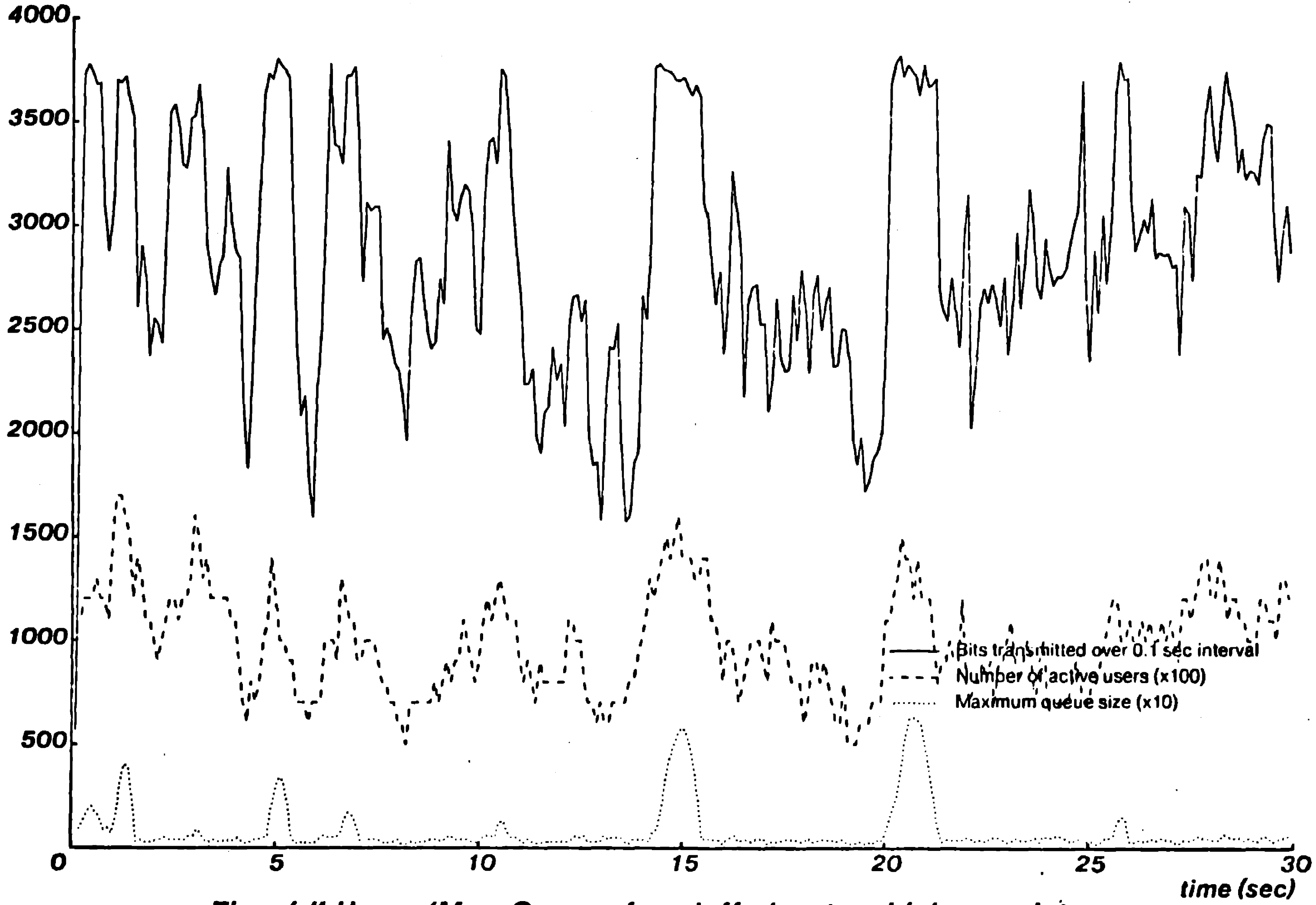
Flow/# Users/Max. Queue for Jaffe/no protocol/slow updates Link 2 (dynamic network)

Figure 5.19



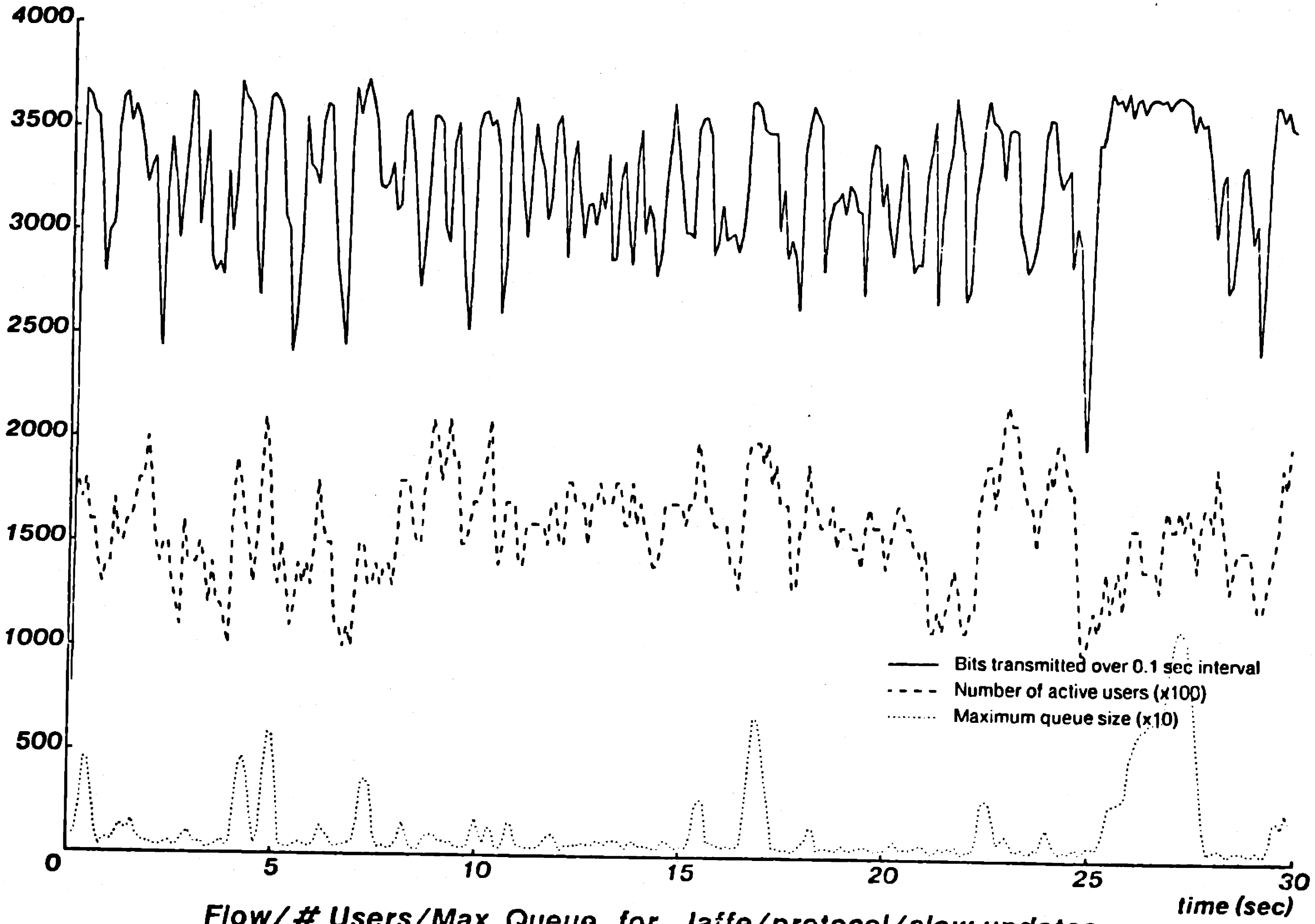
Flow/# Users/Max. Queue for Jaffe/no protocol/slow updates Link 8 (dynamic network)

Figure 5.20



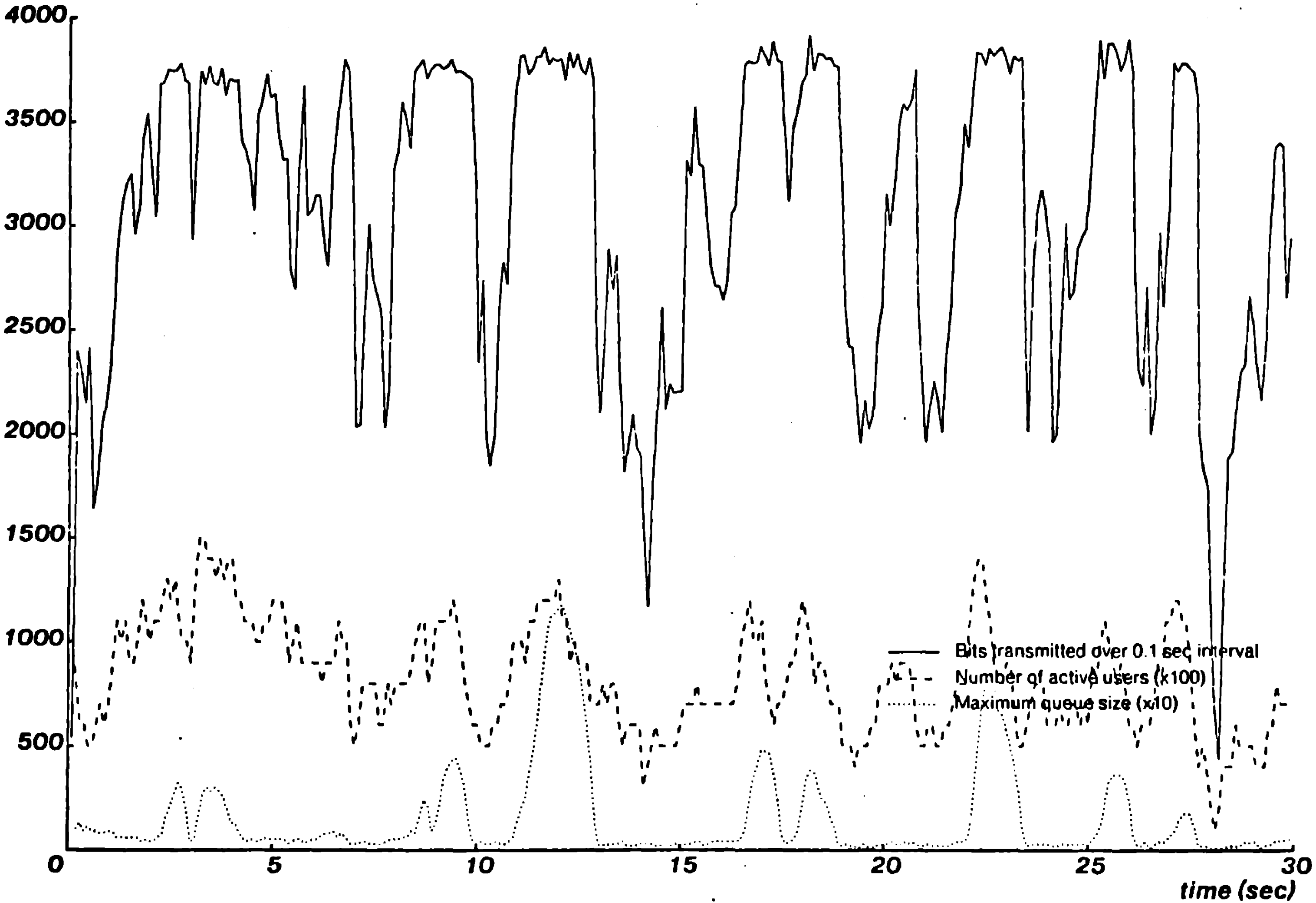
Flow/# Users/Max. Queue for Jaffe/protocol/slow updates
Link 2 (dynamic network)

Figure 5.21



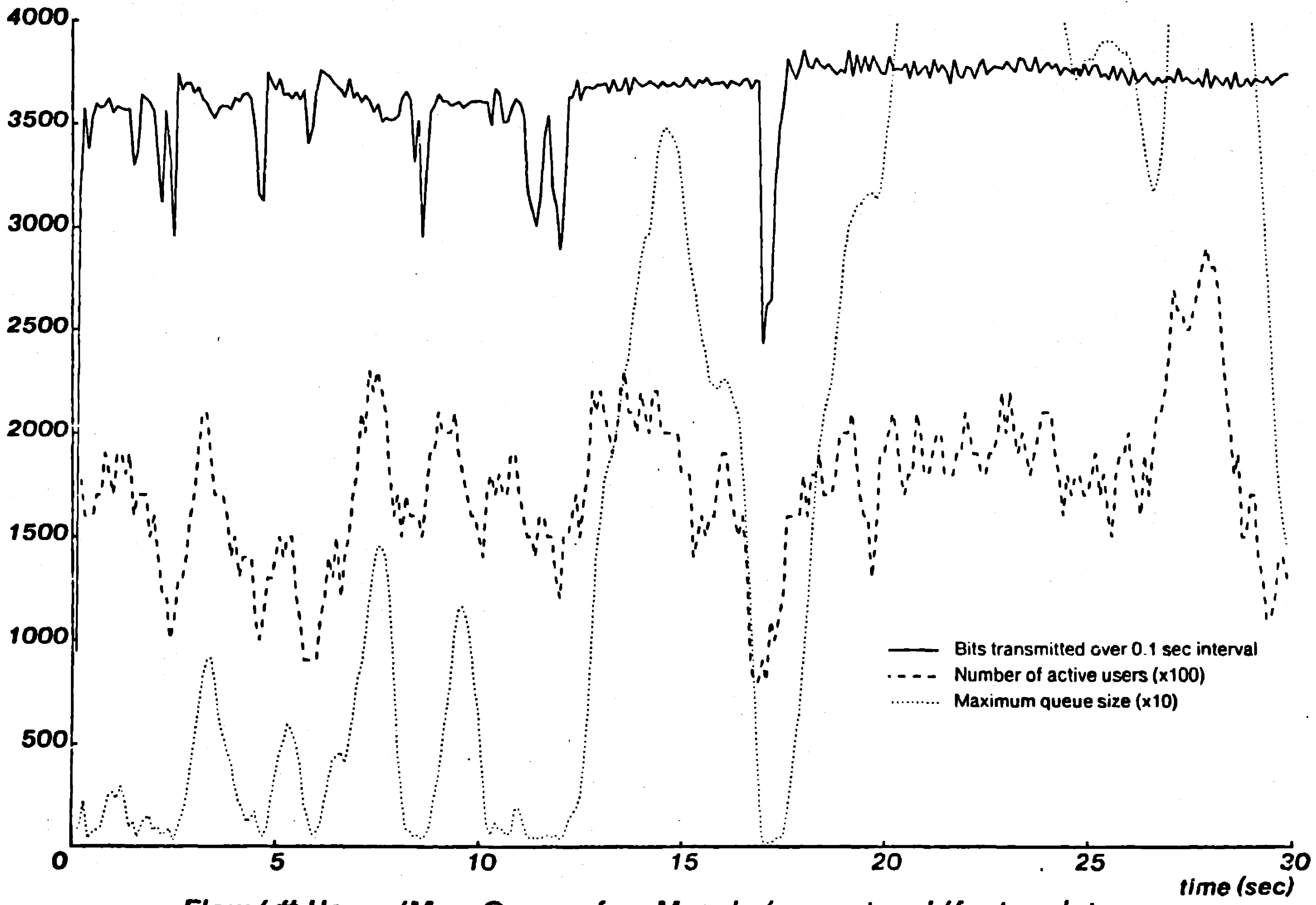
Flow/# Users/Max. Queue for Jaffe/protocol/slow updates
Link 8 (dynamic network)

Figure 5.22



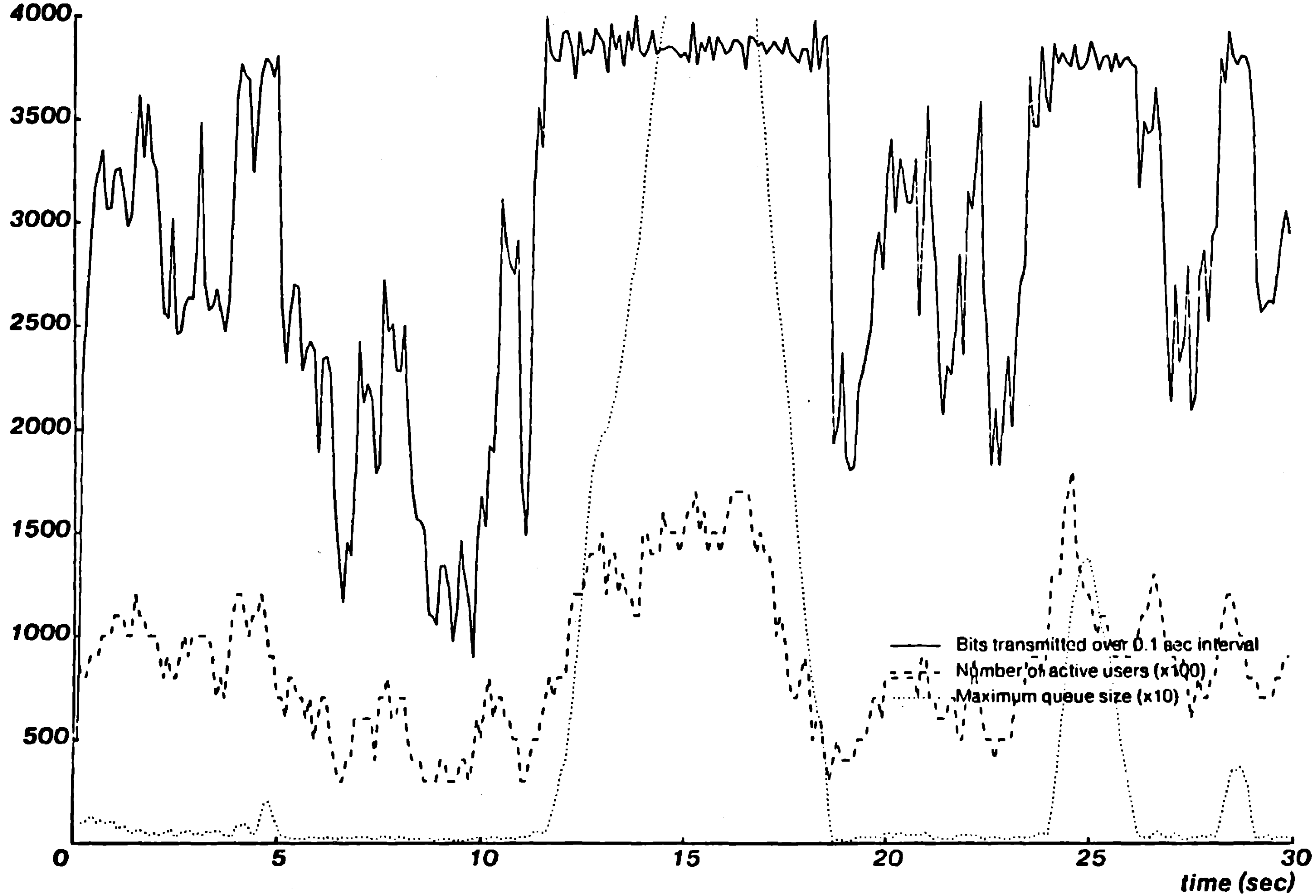
Flow/# Users/Max. Queue for Mosely/no protocol/fast updates
Link 2 (dynamic network)

Figure 5.23



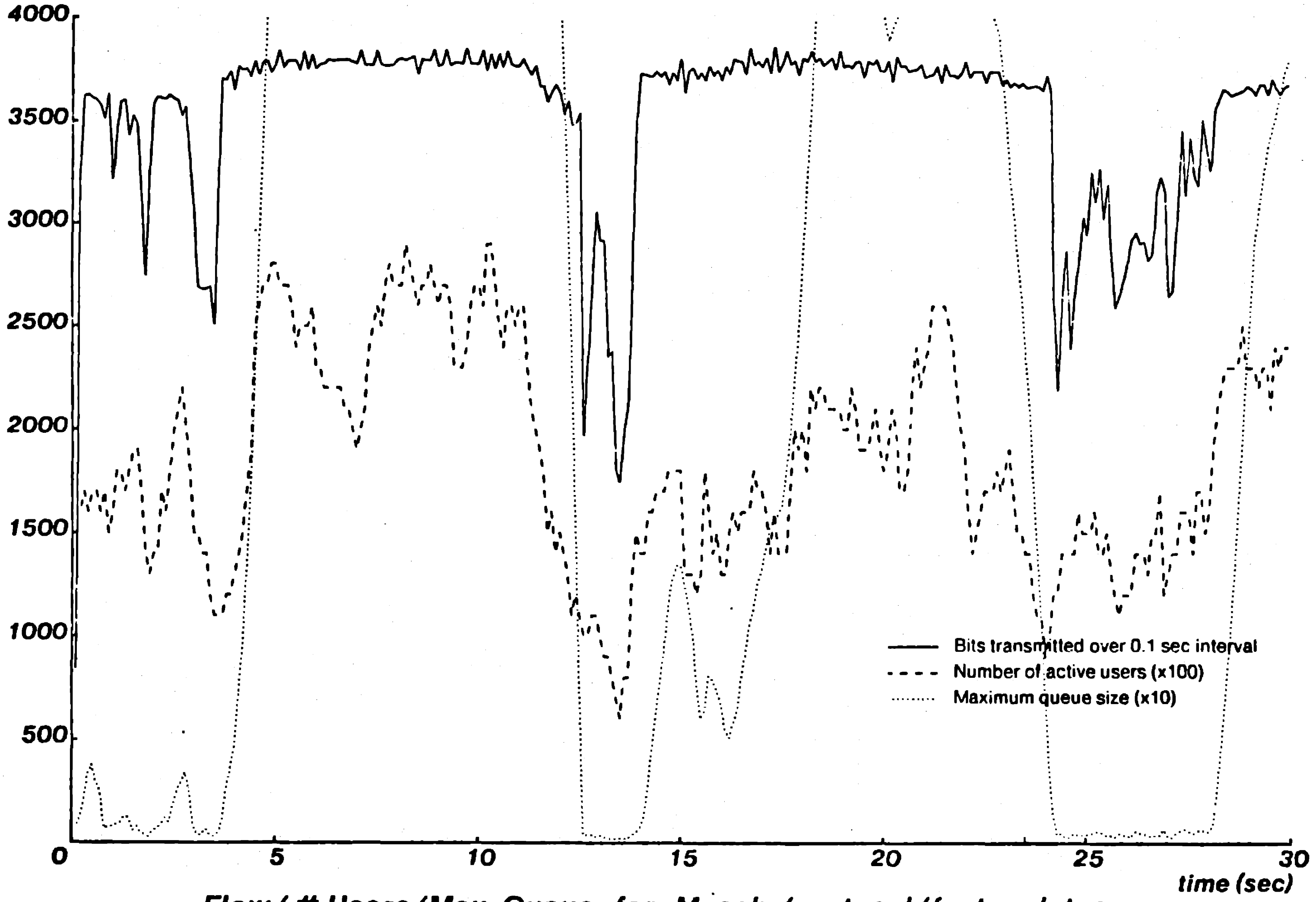
Flow/# Users/Max. Queue for Mosely/no protocol/fast updates
Link 8 (dynamic network)

Figure 5.24



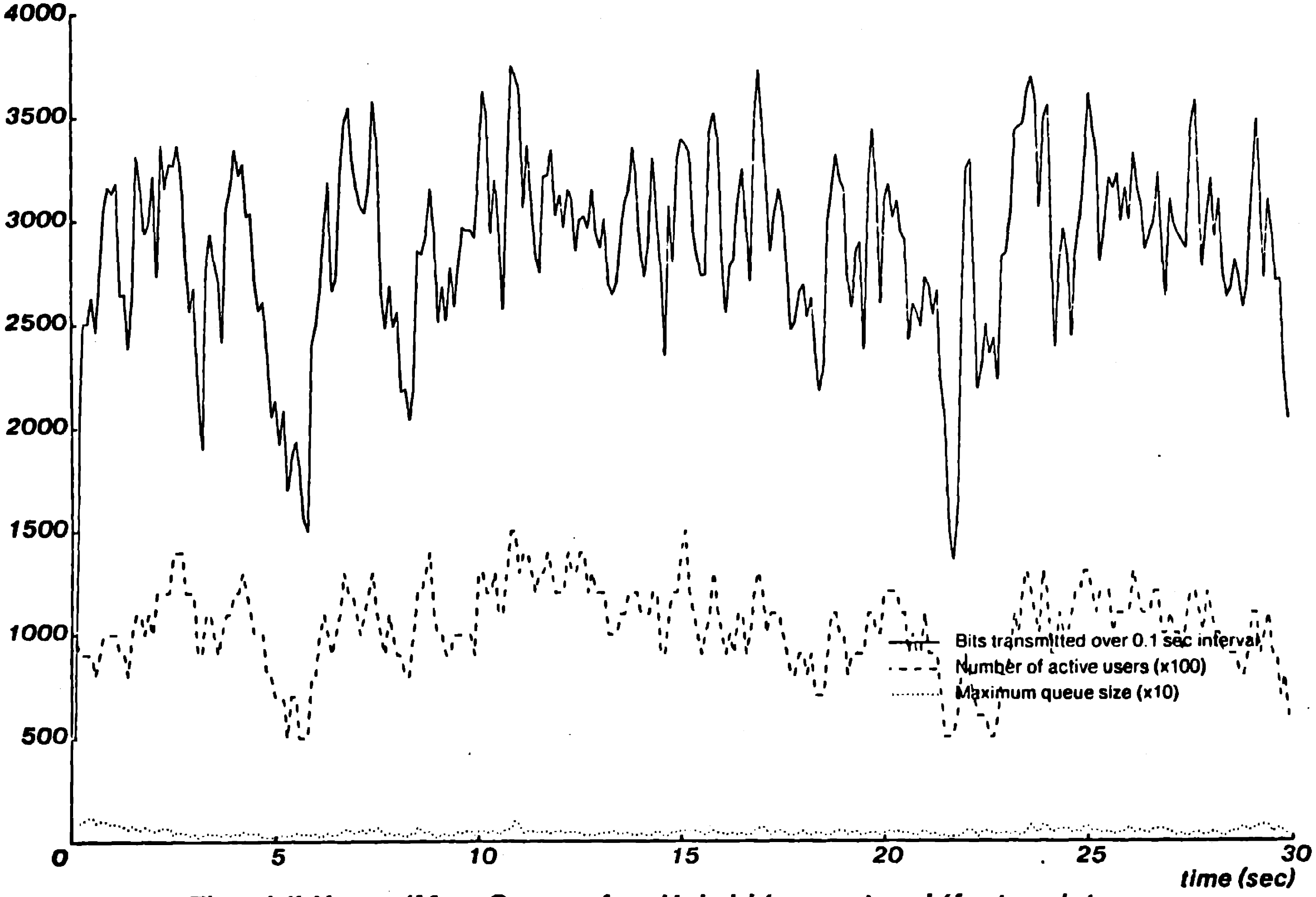
Flow/# Users/Max. Queue for Mosely/protocol/fast updates
Link 2 (dynamic network)

Figure 5.25



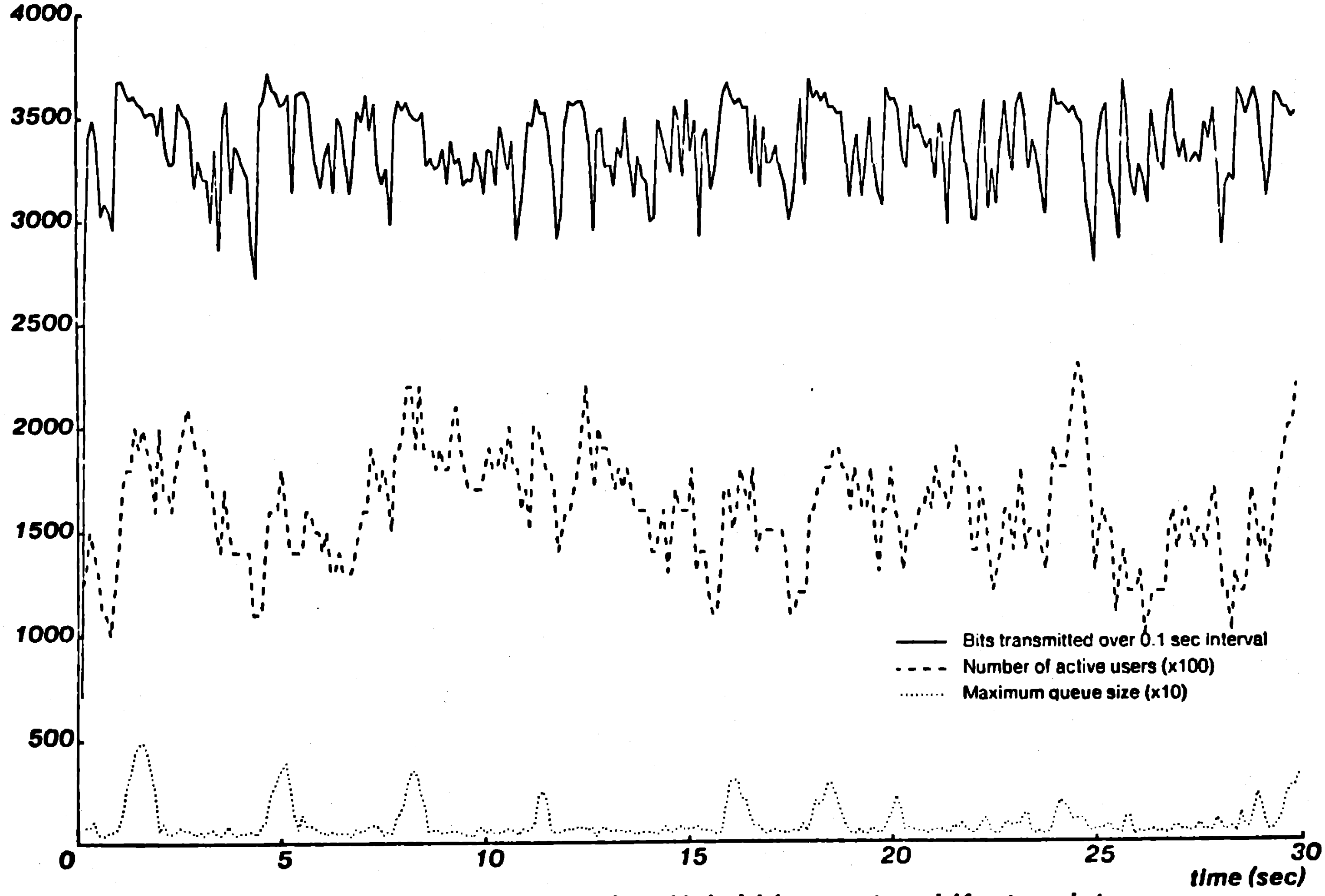
Flow/# Users/Max. Queue for Mosely/protocol/fast updates
Link 8 (dynamic network)

Figure 5.26



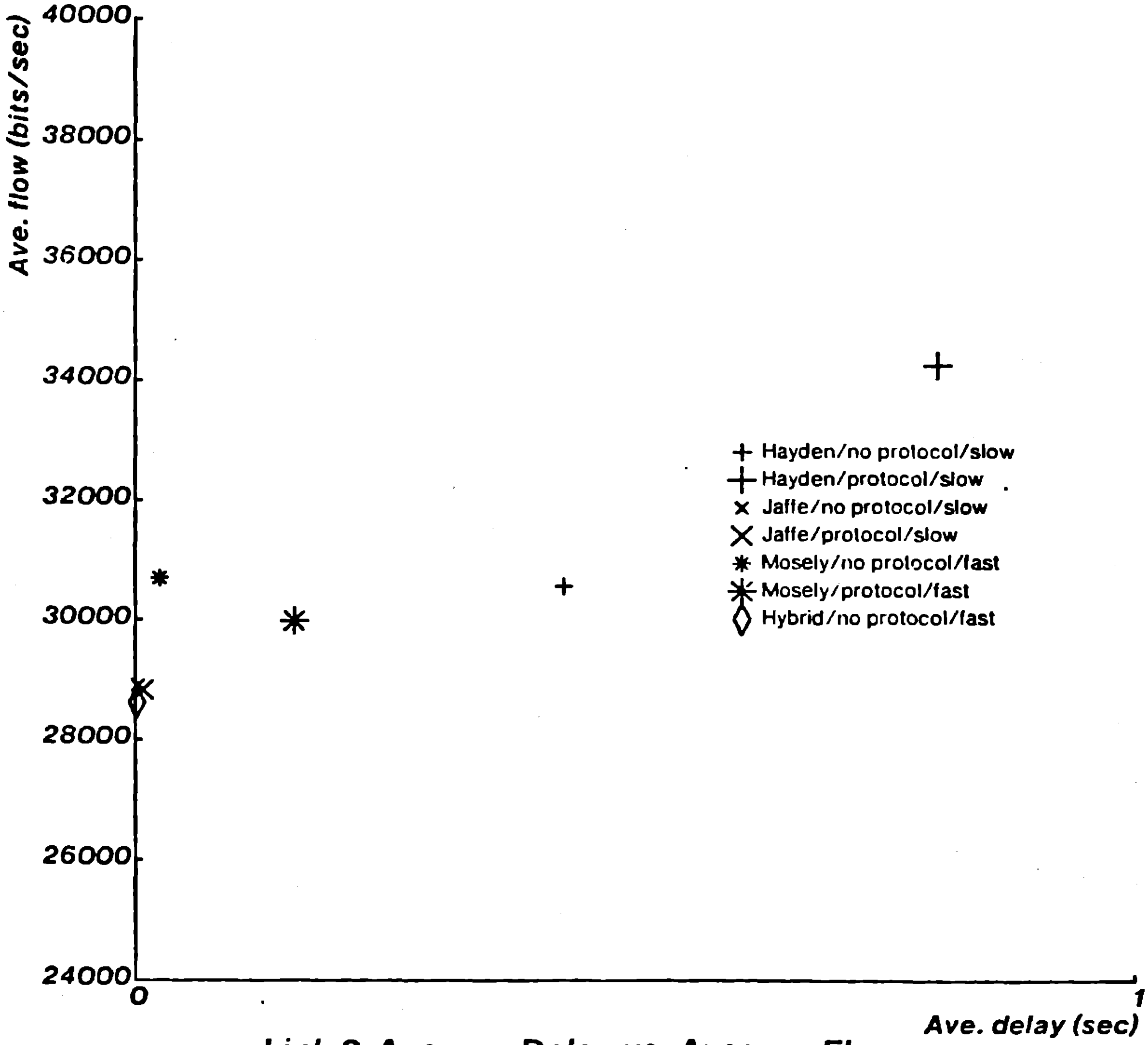
Flow/# Users/Max. Queue for Hybrid/no protocol/fast updates
Link 2 (dynamic network)

Figure 5.27

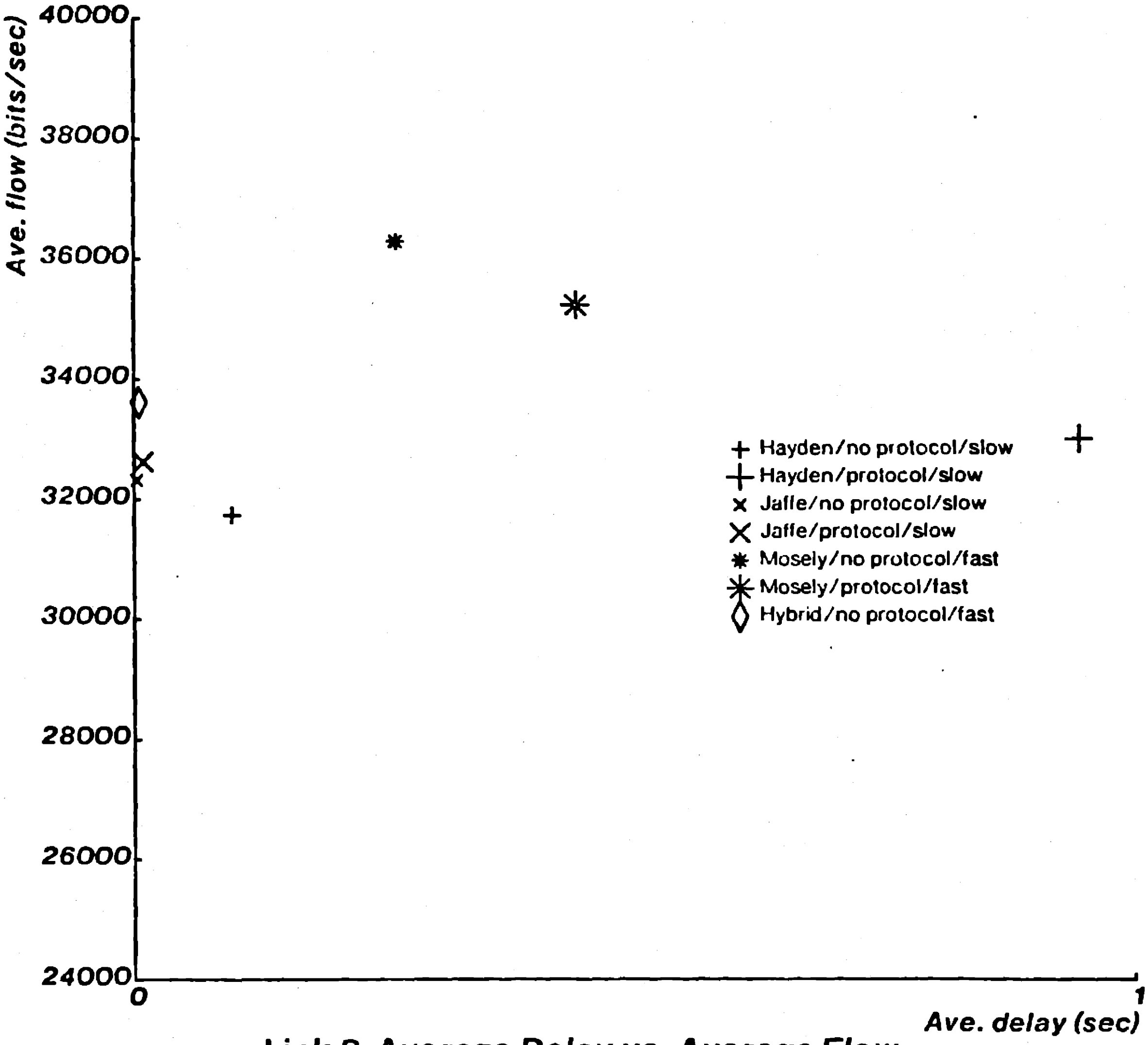


Flow/# Users/Max. Queue for Hybrid/no protocol/fast updates Link 8 (dynamic network)

Figure 5.28



Link 2 Average Delay vs. Average Flow
Figure 5.29



Link 8 Average Delay vs. Average Flow
Figure 5.30

Chapter 6

Suggestions for Further Research

In this thesis, we have considered the problem of designing a distributed fair flow control algorithm that can be implemented asynchronously and remain stable in the presence of feedback delays. After developing an appropriate system model, we analyzed several flow control algorithms. We discovered that one algorithm which is unstable when implemented asynchronously, can be made stable by the addition of an update protocol (the generalized link memory algorithm). One algorithm cannot be made stable even with the update protocol (Hayden), and another is stable even without the update protocol (modified Hayden). This last algorithm is particularly interesting in that the links, when updating their controls, make more use of the available feedback information than the other algorithms.

While the theoretical results indicate that these algorithms should perform well, the results are only valid for a static network. Computer simulations indicate that none of the algorithms can respond to changing input conditions fast enough to effectively control a dynamic network. Therefore, it is necessary to improve the response time of the system. This might be accomplished in several ways. First, queueing priority should be given to the control packets, to speed up convergence of the controls. Also, when links become very congested, we might allow the system to discard low priority packets. Finally, as described at the end of Chapter 5, we could try to devise better update functions that make more complete use of the available feedback information.

As an example, we propose the following update function:

$$p_j^{n+1}=\begin{cases}\max\limits_{i\in\mathcal{S}_j} r_{ij}(t_j^n)+(c_j-f_j^n)/W_j(a) & \text{when } f_j^n\leq c_j\\ F(r_j(t_j^n),c_j) & \text{when } f_j^n>c_j\end{cases}\qquad(6.1)$$

where a is some appropriately chosen constant such that $0\leq a<1$, $W_j(a)$ is the number of sessions whose rate is higher than a max $r_{ij}(t_j^n)$, $r_j(t_j^n)$ is the vector of rates $(\ldots,r_{ij}(t_j^n),\ldots)$, $i\in\mathcal{S}_j$, and $F(R_j,c_j)$ is the maximum of the coordinates of the fair allocation over the set defined by

$$r_{ij}\leq R_{ij}\qquad(6.2.1)$$

and

$$\sum_{i\in\mathcal{S}_j} r_{ij}\leq c_j.\qquad(6.2.2)$$

The idea behind this update function is simple. All the update functions given previously change the controls by conservative amounts, assuming at all times that the links control all their users. This update function lets the link use its knowledge of the rate vector to make better estimates of the number of sessions that are actually under its control. When the flow is less than capacity, the link uses $W_j(a)$ as an estimate of the number of sessions it is currently controlling. When the flow is greater than capacity, the link adopts the largest control that would let each of its sessions send at the minimum of its former rate and the new control, while guaranteeing that the new flow would be less than or equal to capacity.

In conclusion, we remark that flow control algorithms are just one example of many different distributed asynchronous problems. The techniques described in this thesis, that is, the use of update protocols and more complete use of feedback information, might be profitably applied to other problems as well, and the theorems in Chapter 4 might be used to analyze such algorithms

Appendix A

Theorem 3.1. If $d_{ij}(t)$ and $D_{ij}(t)$ satisfy (3.10) and (3.11), then for any t^0 there exists $t^1 \geq t^0$ such that for all $t \geq t^1$,

$$t-d_{ij}(t)-D_{ik}(t-d_{ij}(t)) \geq t^0. \tag{A.1}$$

for all $j \in \mathcal{L}$, $i \in \mathcal{S}_j$, $k \in \mathcal{L}_i$.

Proof. For any t^0, by (3.11) there must exist T_{ij} such that

$$T_{ij}-D_{ij}(T_{ij}) \geq t^0. \tag{A.2}$$

Let $T_i = \max_{j \in \mathcal{S}_i} T_{ij}$. By (3.11), there must also exist t_{ij} such that

$$t_{ij}-d_{ij}(t_{ij}) > T_i. \tag{A.3}$$

Let $t^1 = \max_{j \in \mathcal{L}, i \in \mathcal{S}_j} t_{ij}$. Then for $t \geq t^1$, by (3.10),

$$\begin{aligned} t&-d_{ij}(t)-D_{ik}(t-d_{ij}(t)) \\ &\geq t_{ij}-d_{ij}(t_{ij})-D_{ik}(t_{ij}-d_{ij}(t_{ij})) \\ &\geq T_i-D_{ik}(T_i) \\ &\geq T_{ik}-D_{ik}(T_{ik}) \\ &\geq t^0. \end{aligned} \tag{A.4}$$

This completes the proof of Theorem 3.2.

Appendix B

Lemma 3.2.1. Let $F_j(\underline{z})=(\min_{k} z_k)(1-W_j/(2\hat{W}_j))+c_j/(2\hat{W}_j)$ and define $F(\underline{z})=(\ldots,F_j(\underline{z}),\ldots)$. Let $z^0=\underline{0}$ and define $\underline{z}^{n+1}=\underline{F}(z^n)$. Then $z_j^n \to z_j{*}=p^1+W_j/(2\hat{W}_j)(c_j/W_j-p^1)$ and $z_j^n<c_j/W_j$ for all n.

Proof. First we show that $\underline{z}*$ is a fixed point of $F_j(\cdot)$. Note that since $p^1=\min_{j} c_j/W_j$, $z_j{*}\geq p^1$ for all j and $p^1=\min_{k} z_k{*}$. Hence

$$\begin{aligned} F_j(z_j{*}) &= p^1(1-W_j/(2\hat{W}_j))+c_j/(2\hat{W}_j) \\ &= z_j{*}. \end{aligned} \tag{B.1}$$

Now let $||\underline{z}||=\max_{j}|z_j|$. We show that $F(\cdot)$ is a contraction under this norm.

$$\begin{aligned} ||F(\underline{x})-F(\underline{y})|| &= \max_{j} |(1-W_j/(2\hat{W}_j))(\min_{k} x_k-\min_{k} y_k)| \\ &= \max_{j} (1-W_j/(2\hat{W}_j))|(\min_{k} x_k-\min_{k} y_k)| \\ &\leq (1-\min_{j} W_j/(2\hat{W}_j))\max_{k}|x_k-y_k| \\ &= (1-\min W_j/(2\hat{W}_j))||\underline{x}-\underline{y}||. \end{aligned} \tag{B.2}$$

Hence, $z^n \to \underline{z}*$.

Now suppose $z_j^n<c_j/W_j$ for each j, for some n. Then

$$\begin{aligned} z_j^{n+1} &< (1-W_j/(2\hat{W}_j))c_j/W_j+c_j/(2\hat{W}_j) \\ &= c_j/W_j. \end{aligned} \tag{B.3}$$

Hence, by induction, $z_j^n<c_j/W_j$ for all n. This completes the proof of Lemma 3.2.1.

Appendix C

Theorem 3.4. Let S be a linear space with norm $||\cdot||$ such that $\{x: ||x||\leq c\}$ is compact for all c. Let $f:S\rightarrow S$ and $f_n:S\rightarrow S$ be functions such that $f_n\rightarrow f$ uniformly, and such that $||f(x) - f(y)||<||x - y||$ for all $x,y\in S$. Suppose there exists x^* such that $x^*=f(x^*)$. Define $x_{n+1}=f_n(x_n)$. Then $x_n\rightarrow x^*$.

Proof. Without loss of generality, we assume $x^*=0$. For each $e>0$, define

$$d(e)=\max_{||x||\leq e} ||f(x)||. \tag{C.1}$$

The maximum must exist because $\{x: ||x||\leq c\}$ is compact for all c. Note that $d(e)<e$, since $||f(x)||<||x||$.

Now,

$$\begin{aligned}||f(x)-f(e(x/||x||))||&<||x-e(x||x||)||\\ &\leq||x||-e\end{aligned} \tag{C.2}$$

and

$$\begin{aligned}||f(x)||&<||x||-e+f(e(x/||x||))\\ &\leq||x||-(e-d(e)).\end{aligned} \tag{C.3}$$

Now let $e_1=e/2$ and find N such that

$$||f_n(x)-f(x)||\leq(e_1-d(e_1))/2 \tag{C.4}$$

for all $n\geq N$. Then

$$
\begin{aligned}
||x_n|| &= ||f_n(x_{n-1})|| \\
&\leq ||f(x_{n-1})|| + (e_1 - d(e_1))/2 \\
&< \max\,(e_1, ||x_{n-1}|| - (e_1 - d(e_1))) + (e_1 - d(e_1))/2 \\
&< \max\,(e_1, ||x_{n-1}|| - (e_1 - d(e_1))/2), \qquad \text{(C.5)}
\end{aligned}
$$

where the third step follows by (C.3).

So for any x_0,

$$||x_1|| < \max\,(e, ||x_0|| - (e_1 - d(e_1))/2), \qquad \text{(C.6)}$$

and by induction on n,

$$||x_n|| < \max\,(e, ||x_0|| - n(e_1 - d(e_1))/2). \qquad \text{(C.7)}$$

Thus, for any $e>0$, if $n \geq 2||x_0||/(e_1 - d(e_1))$, $||x_n||<e$. This completes the proof of Theorem 3.4

Appendix D

Define, for $k=1,\ldots,L$,

$$e^k(0)= aE \tag{D.1.1}$$

and

$$E^k(0)= E \tag{D.1.2}$$

where $a= \min_{j\in\mathcal{L}} 1/W_j$ and E is some suitably large constant such that $E\geq p^L/a$. Let

$$e^1(n)=0 \tag{D.2.1}$$

and

$$E^1(n+1)=(1-a)E \tag{D.2.2}$$

for all $n\geq1$. Now define, for $n\geq N_k$

$$e^k(n+1)=(1-a)(e^k(n)+E^{k-1}(n)) \tag{D.3.1}$$

$$E^k(n+1)=(1-a)(e^k(n)+E^k(n)) \tag{D.3.2}$$

and for $N_k\leq n<N_{k+1}$, $K>k$,

$$e^K(n+1)= p^K-p^k+e^k(n+1) \tag{D.4.1}$$

$$E^K(n+1)=E, \tag{D.4.2}$$

where $N_1<N_2<\ldots<N_L$, $N_1=1$ and, for $k>1$,

$$e^{k-1}(n)+E^{k-1}(n)<(p^k-p^{k-1})a/(1-a). \tag{D.5}$$

for all $n\geq N_k$.

We show by induction that the sequence $N_1<N_2<\ldots<N_L$ satisfying (D.5) exists, and that the sequences $\{e^k(n)\}$ and $\{E^k(n)\}$ are monotonically non-increasing and converge to 0, for all k.

Furthermore, we show that the sequences have the properties that, for k<K, for all n,

$$e^k(n)\leq e^K(n) \tag{D.6.1}$$

$$E^k(n)\leq E^K(n) \tag{D.6.2}$$

and

$$p^k-e^k(n)\leq p^K-e^K(n). \tag{D.6.3}$$

By (D.2), $e^1(n)=0$ and $E^1(n)\rightarrow 0$. Hence, there must exist a time N_2 such that for all $n\geq N_2$,

$$e^1(n)+E^1(n)<(p^2-p^1)a/(1-a). \tag{D.7}$$

Suppose we have $\{e^{K-1}(n)\}$ and $\{E^{K-1}(n)\}$ such that $e^{K-1}(n)\rightarrow 0$ and $E^{K-1}(n)\rightarrow 0$. Then there must exist a time N_K such that, for all $n\geq N_K$

$$e^{K-1}(n)+E^{K-1}(n)<(p^K-p^{K-1})a/(1-a). \tag{D.8}$$

Now for $n\geq N_K$,

$$e^K(n+1)=(1-a)(e^K(n)+E^{K-1}(n)). \tag{D.9}$$

Since $(1-a)<1$ and $E^{K-1}(n)\rightarrow 0$, $e^K(n)\rightarrow 0$. Similarly, $e^K(n)\rightarrow 0$ implies $E^K(n)\rightarrow 0$. Hence, we have shown by induction, the existence of $N_1<N_2<\ldots<N_L$, and that $e^k(n)\rightarrow 0$ and $E^k(n)\rightarrow 0$ for each k.

We show by induction that $e^K(n)$ and $E^K(n)$ are monotonically non-increasing for each k. Clearly $e^1(n)$ and $E^1(n)$ are monotonically non-increasing. Now suppose that $e^{K-1}(n)$ and $E^{K-1}(n)$ are monotonic non-increasing sequences. Then $e^K(n+1)\leq e^K(n)$ and $E^K(n+1)\leq E^K(n)$ for $n<N_K$. For $n=N_K$,

$$e^K(N_{K+1})=(1-a)(e^K(N_K)+E^{K-1}(N_K))$$

$$
\begin{aligned}
&=(1-a)(p^{K}-p^{K-1}+e^{K-1}(N_{K})+E^{K-1}(N_{K}))\\
&<(1-a)(p^{K}-p^{K-1})+a(p^{K}-p^{K-1})\\
&=p^{K}-p^{K-1}\\
&<e^{K}(N_{K}). \qquad (D.10)
\end{aligned}
$$

Also,

$$
\begin{aligned}
E^{K}(N_{K+1})&=(1-a)(e^{K}(N_{K})+E^{K}(N_{K}))\\
&\leq(1-a)(e^{K}(1)+E^{K}(1))\\
&=(1-a)(p^{K}-p^{1}+E)\\
&<(1-a)(p^{L}+E)\\
&<E\\
&=E^{K}(N_{K}). \qquad (D.11)
\end{aligned}
$$

The next to last step holds because E was chosen greater than or equal to p^{L}/a.

Now suppose $e^{K}(n)<e^{K}(n-1)$ for some $n\geq N_{K+1}$. Then

$$
\begin{aligned}
e^{K}(n+1)&=(1-a)(e^{K}(n)+E^{K-1}(n))\\
&\leq(1-a)(e^{K}(n-1)+E^{K-1}(n-1))\\
&=e^{K}(n). \qquad (D.12)
\end{aligned}
$$

Similarly, if $E^{K}(n)<E^{K}(n-1)$,

$$
\begin{aligned}
E^{K}(n+1)&=(1-a)(e^{K}(n)+E^{K}(n))\\
&\leq(1-a)(e^{K}(n-1)+E^{K}(n-1))\\
&=E^{K}(n). \qquad (D.13)
\end{aligned}
$$

Hence, $e^{k}(n)$ and $E^{k}(n)$ are monotonically non-increasing for all k.

Next we show that (D.6) holds. For $n=0$, $e^{k}(0)=aE$ and $E^{k}(0)=E$ for all k. For $n=1$, $e^{k}(1)=p^{k}-p^{1}$ and $E^{k}(1)=E$. Hence, (D.6) holds for $n=0$ and $n=1$.

We show that (D.6.1) and (D.6.2) hold by induction. Suppose that (D.6.1) and (D.6.2) are satisfied for some $n\geq 1$. Let J be the largest number such that $n\geq N_J$. We show that (D.6.1) and (D.6.2) are satisfied for $n+1$, for $k<K$. We consider three cases: $k<K\leq J$, $k\leq J<K$, $J<k<K$.

Let $k<K\leq J$. Then

$$\begin{aligned} e^k(n+1)&=(1-a)(e^k(n)+E^{k-1}(n))\\ &\leq(1-a)(e^K(n)+E^{K-1}(n))\\ &=e^K(n+1) \end{aligned} \tag{D.14}$$

and

$$\begin{aligned} E^k(n+1)&=(1-a)(e^k(n)+E^k(n))\\ &\leq(1-a)(e^K(n)+E^K(n))\\ &=E^K(n+1). \end{aligned} \tag{D.15}$$

Let $k\leq J<K$. Then by (D.14),

$$\begin{aligned} e^k(n+1)&\leq e^J(n+1)\\ &\leq p^K-p^J+e^J(n+1)\\ &=e^K(n+1) \end{aligned} \tag{D.16}$$

and

$$\begin{aligned} E^k(n+1)&\leq E^J(n+1)\\ &\leq E\\ &=E^K(n+1). \end{aligned} \tag{D.17}$$

Finally, suppose $J<k<K$. Then

$$e^k(n+1)=p^k-p^J+e^J(n+1)$$

$$\leq p^K - p^J + e^J(n+1)$$
$$= e^K(n+1) \qquad (D.18)$$

and

$$E^k(n+1) = E = E^K(n+1). \qquad (D.19)$$

So, by induction, (D.6.1) and (D.6.2) hold for all n.

Finally, we show that (D.6.3) holds for all $n \geq 1$. Let J be defined as above. We have already shown in (D.10), for $n \geq N_k$

$$e^k(n) \leq e^k(n+1)$$
$$< p^k - p^{k-1}. \qquad (D.20)$$

Thus, for $K \leq J$,

$$p^k - e^k(n) \leq p^k$$
$$\leq p^{K-1}$$
$$< p^K - e^K(n). \qquad (D.21)$$

For $J<K$,

$$p^k - e^k(n) \leq p^J - e^J(n)$$
$$= p^K - e^K(n). \qquad (D.22)$$

Therefore, (D.6.3) holds for all n.

This completes Appendix D.

Appendix E

```
;;; -*- Mode:LISP; Package:USER; Base:10; Fonts:MEDFNT -*-

;;; **************************************************
;;; Copyright (c) 1984 by
;;; Jeannine Mosely and Allan C. Wechsler
;;;
;;; It is the intention of the authors that this
;;; software remain in the public domain, and that
;;; no one shall impede its distribution, nor
;;; distribute it for profit.
;;; **************************************************

(DEFSTRUCT (USER :CONC-NAME)
  PARTNER                                 ; Another user.
  ROUTE                                   ; A list of links.
  (RATE 500)                              ; Bits per second.
  ID                                      ; A number.
  (PARTNER-RATE 1000)                     ; Bits per second.
  ;; The following six components are statistics that we
  ;; reset every after statistics collection.
  (TOTAL-VOICE-PACKET-DELAY 0)
  (TOTAL-CONTROL-PACKET-DELAY 0)
  (NUMBER-OF-VOICE-PACKETS 0)
  (NUMBER-OF-CONTROL-PACKETS 0)
  (MAX-VOICE-PACKET-DELAY 0)
  (MAX-CONTROL-PACKET-DELAY 0)
  TALKING?                                ; T or NIL.
  PRINT-STATISTICS?                       ; T or NIL.
  )

(DEFSTRUCT (LINK :CONC-NAME)
  USERS                                   ; An a-list of
                                          ; active users and
                                          ; their rates.

  (NUMBER-OF-USERS 0)
  QUEUE-FRONT                             ; A list of packets.
  QUEUE-BACK                              ; The last vertebra
                                          ; of QUEUE-FRONT.
                                          ; (Efficiency hack.)
  (QUEUE-LENGTH 0)
  ;; Max queue length is reset after each statistics
  ;; collection.
  (MAX-QUEUE-LENGTH 0)
  (CONTROL 1000)                          ; Bits per second.
  (CAPACITY 40000)                        ; Bits per second.
  (PROPAGATION-DELAY 0.003)               ; Seconds.
  PACKET-NOW-TRANSMITTING                 ; A packet, or NIL
                                          ; if link idle.

  (NUMBER-OF-PACKETS-SENT 0)              ; per stats.
  (TOTAL-NUMBER-OF-PACKETS-SENT 0)        ; ever.
  (NUMBER-OF-BITS-SENT 0)                        ; per stats.
  (TOTAL-NUMBER-OF-BITS-SENT 0)                  ; ever.
  (TOTAL-PACKET-DELAY 0)                         ; ever.
  (TOTAL-SQUARED-PACKET-DELAY 0)                 ; ever.
  (PACKET-DELAY-HISTOGRAM                        ; ever.
    (MAKE-ARRAY 11 ':TYPE 'ART-16B))
  (PRINT-STATISTICS? T)                          ; T or NIL
  ID)                                            ; A number.

(DEFSTRUCT (PACKET :CONC-NAME)
  (FORWARD-CONTROL 1000000)                      ; Minimum control
                                                 ; seen so far on
                                                 ; this traverse.
  FEEDBACK-CONTROL                               ; Control data
                                                 ; going back to
                                                 ; starting point.
  LENGTH
  ROUTE
  GENERATION-TIME
  ARRIVAL-TIME
  LAST-IN-TALK-SPURT?
  TYPE                                           ; VOICE or CONTROL.
  SOURCE
  SOURCE-RATE
  DESTINATION)

(DEFSTRUCT (EVENT :CONC-NAME)
  FUNCTION
  TIME
  ARGUMENTS)

;;; Global variables.

(DECLARE (SPECIAL *USERS*                        ; All users.
                  *LINKS*                        ; All links.
                  *TIME*                         ; Simulated.
                  *USER-STAT-STREAM*
                  *LINK-STAT-STREAM*
                  *EVENT-TABLE*                  ; The Heap of
                                                 ; Things to Come.
                  *NEXT-EVENT-NUMBER*))          ; Index into heap.

(DEFCONST *PACKET-ABSORPTION-DELAY* 0.0005)
(DEFCONST *PACKET-ARRIVAL-DELAY* 0.0005)
(DEFCONST *PACKET-TRANSMISSION-DELAY* 0.0001)
(DEFCONST *PACKET-GENERATION-DELAY* 0.0005)

(DEFCONST *AVERAGE-TALK-SPURT-LENGTH* 60)        ; In packets.
(DEFCONST *USERS-TALK-FOREVER* T)                ; Infinite talk
                                                 ; spurts?  T or NIL.
(DEFCONST *CONTROL-PACKET-SPACING* 0.10)         ; Seconds.

(DEFCONST *UPDATE-INTERVAL* 0.10)                ; Seconds.
```

```
(DEFCONST *UPDATE-ATTEMPT-INTERVAL* 0.02) ; Seconds.

(DEFCONST *UPDATE-PROTOCOL* 'MOSELY)      ; HAYDEN or MOSELY.
(DEFCONST *UPDATE-FUNCTION* 'MOSELY-UPDATE-FUNCTION)

(DEFCONST *LINK-STATISTICS-INTERVAL* 0.1)
(DEFCONST *USER-STATISTICS-INTERVAL* 0.5)

(DEFCONST *USERS-TO-PRINT*
  '(1 9 15 19 21 29 35 39 41 49 55 59 61 69 75 79))


;;; Top level function.

(DEFUN RUN-NETWORK (TIME-LIMIT)
  (INITIALIZE)
  (WITH-OPEN-FILE
   (*USER-STAT-STREAM* "oz:<j9>user-stats.text" ':OUT)
   (WITH-OPEN-FILE
    (*LINK-STAT-STREAM* "oz:<j9>link-stats.text" ':OUT)
    (FORMAT *USER-STAT-STREAM* "
   User ID      Time       AVD       MVD       ACD
       MCD      Rate   Feedback")
    (FORMAT *LINK-STAT-STREAM* "
   Link ID      Time    # Bits   Control   # Users
    Max. Q         Q  # Packets")
    (SETQ *TIME* -1)
    (SIMULATE TIME-LIMIT)
    (PRINT-LINK-HISTOGRAMS))))

;;; Network initialization.

(DEFUN INITIALIZE ()
  (SETQ *USERS* NIL)
  (CLEAR-EVENT-TABLE)
  (LET ((LINK-1 (MAKE-LINK ID 1))
        (LINK-2 (MAKE-LINK ID 2 PRINT-STATISTICS? T))
        (LINK-3 (MAKE-LINK ID 3))
        (LINK-4 (MAKE-LINK ID 4))
        (LINK-5 (MAKE-LINK ID 5))
        (LINK-6 (MAKE-LINK ID 6))
        (LINK-7 (MAKE-LINK ID 7))
        (LINK-8 (MAKE-LINK ID 8 PRINT-STATISTICS? T)))
    (SETQ *LINKS* (LIST LINK-1 LINK-2 LINK-3 LINK-4
                        LINK-5 LINK-6 LINK-7 LINK-8))
    (USERS '((1 8) (21 28) (41 48) (61 68)) LINK-8)
    (USERS '((9 14) (29 34) (49 54) (69 74)) LINK-7)
    (USERS '((15 18) (35 38) (55 58) (75 78)) LINK-6)
    (USERS '((19 20) (39 40) (59 60) (79 80)) LINK-5)

    (USERS '((1 20)) LINK-1)
    (USERS '((21 40)) LINK-2)
    (USERS '((41 60)) LINK-3)
    (USERS '((61 80)) LINK-4))
  (ESTABLISH-USER-PARTNERS)
  (INITIALIZE-USERS-TO-PRINT))

(DEFUN INITIALIZE-USERS-TO-PRINT ()
  (DOLIST (USER-TO-PRINT *USERS-TO-PRINT*)
    (SETF (USER-PRINT-STATISTICS?
            (FIND-KNOWN-USER USER-TO-PRINT)) T)))

(DEFUN USERS (ID-RANGES LINK)
  (LOOP FOR (LOW-ID HIGH-ID) IN ID-RANGES
        DO
        (LOOP FOR ID FROM LOW-ID TO HIGH-ID
              DO
              (ADD-OR-MODIFY-USER ID LINK))))

(DEFUN ADD-OR-MODIFY-USER (ID LINK)
  (LET ((USER (FIND-USER ID)))
    (PUSH LINK (USER-ROUTE USER))))

(DEFUN ESTABLISH-USER-PARTNERS ()
  (LOOP FOR USER IN *USERS*
        DO
        (SETF (USER-PARTNER USER)
              (FIND-KNOWN-USER (- 81 (USER-ID USER))))))

(DEFUN FIND-USER (ID)
  (OR (FIND-KNOWN-USER ID)
      (LET ((USER (MAKE-USER ID ID)))
        (PUSH USER *USERS*)
        USER)))

(DEFUN FIND-KNOWN-USER (ID)
  (LOOP FOR USER IN *USERS*
        DO
        (WHEN (= (USER-ID USER) ID)
          (RETURN USER))))

;;; Event table hackery.

(DEFUN CLEAR-EVENT-TABLE ()
  (SETQ *EVENT-TABLE* (MAKE-ARRAY 2048))
  (SETQ *NEXT-EVENT-NUMBER* 1))

(DEFUN ADD-EVENT-TO-HEAP (EVENT)
  (PERCOLATE-UP EVENT *NEXT-EVENT-NUMBER*)
  (INCF *NEXT-EVENT-NUMBER*))

(DEFUN GET-NEXT-EVENT ()
  (WHEN (> *NEXT-EVENT-NUMBER* 1)
    (LET ((EVENT (AREF *EVENT-TABLE* 1))
          (HOLE (PERCOLATE-DOWN 1)))
      (UNLESS (= HOLE (- *NEXT-EVENT-NUMBER* 1))
        (PERCOLATE-UP (AREF *EVENT-TABLE*
                            (- *NEXT-EVENT-NUMBER* 1))
                      HOLE))
      (DECF *NEXT-EVENT-NUMBER*)
      EVENT)))

(DEFUN PERCOLATE-UP (EVENT INDEX)
  (LET ((PARENT-INDEX (LSH INDEX -1)))
    (LET ((PARENT-EVENT (AREF *EVENT-TABLE*
                              PARENT-INDEX)))
      (IF (OR (= PARENT-INDEX 0)
              (EVENTS-IN-ORDER PARENT-EVENT
                               EVENT))

          ;; EVENT goes here -- put it here.
          (SETF (AREF *EVENT-TABLE* INDEX) EVENT)

          ;; EVENT goes higher --
          ;; put parent here and recurse.
          (SETF (AREF *EVENT-TABLE* INDEX)
                PARENT-EVENT)
          (PERCOLATE-UP EVENT PARENT-INDEX)))))

(DEFUN PERCOLATE-DOWN (INDEX)
  (LET ((LEFT-CHILD-INDEX (LSH INDEX 1)))
    (IF ( LEFT-CHILD-INDEX *NEXT-EVENT-NUMBER*)
        INDEX
        (LET ((RIGHT-CHILD-INDEX (+ 1 LEFT-CHILD-INDEX))
              (LEFT-CHILD (AREF *EVENT-TABLE*
                                LEFT-CHILD-INDEX)))
          (IF (< RIGHT-CHILD-INDEX *NEXT-EVENT-NUMBER*)
              (LET ((RIGHT-CHILD (AREF *EVENT-TABLE*
                                       RIGHT-CHILD-INDEX)))
                (COND ((EVENTS-IN-ORDER LEFT-CHILD
                                        RIGHT-CHILD)
                       (SETF (AREF *EVENT-TABLE* INDEX)
                             LEFT-CHILD)
                       (PERCOLATE-DOWN LEFT-CHILD-INDEX))
                      (T
                       (SETF (AREF *EVENT-TABLE* INDEX)
                             RIGHT-CHILD)
                       (PERCOLATE-DOWN RIGHT-CHILD-INDEX))))
              (SETF (AREF *EVENT-TABLE* INDEX)
                    LEFT-CHILD)
              LEFT-CHILD-INDEX)))))

(DEFUN EVENTS-IN-ORDER (E1 E2)
  (LET ((T1 (EVENT-TIME E1))
        (T2 (EVENT-TIME E2)))
    (OR
     (< T1 T2)
     (AND
      (= T1 T2)
      (LET ((F1 (EVENT-FUNCTION E1))
            (F2 (EVENT-FUNCTION E2)))
        (OR (AND (EQ F1 #'LINK-STATISTICS-COLLECTION)
                 (NOT (EQ F2 #'LINK-STATISTICS-COLLECTION)))
            (AND (EQ F1 #'LINK-STATISTICS-COLLECTION)
                 (EQ F2 #'LINK-STATISTICS-COLLECTION)
                 (< (LINK-ID (CAR (EVENT-ARGUMENTS E1)))
                    (LINK-ID (CAR (EVENT-ARGUMENTS E2)))))
            (AND (EQ F1 #'USER-STATISTICS-COLLECTION)
                 (NOT (EQ F2 #'USER-STATISTICS-COLLECTION))
                 (NOT (EQ F2
                          #'LINK-STATISTICS-COLLECTION)))
            (AND (EQ F1 #'USER-STATISTICS-COLLECTION)
                 (EQ F2 #'USER-STATISTICS-COLLECTION)
                 (< (USER-ID (CAR (EVENT-ARGUMENTS E1)))
                    (USER-ID (CAR (EVENT-ARGUMENTS E2)))))))
      ))))

;;; The Guts.

(DEFUN SIMULATE (TIME-LIMIT)
  (EVENT #'SIMULATION-STARTUP 0)
  (LOOP FOR EVENT = (GET-NEXT-EVENT)
        WHILE EVENT
        UNTIL (< TIME-LIMIT (EVENT-TIME EVENT))
        DO
        (PERFORM-EVENT EVENT)))

(DEFUN PERFORM-EVENT (EVENT)
  (SETQ *TIME* (EVENT-TIME EVENT))
  (LEXPR-FUNCALL (EVENT-FUNCTION EVENT)
                 (EVENT-ARGUMENTS EVENT)))

;;; *********************************************************
;;;                        Events.
;;; *********************************************************
;;;

;;; Schedule an event

(DEFUN EVENT (EVENT-FUNCTION TIME &REST EVENT-ARGUMENTS)
  (WHEN (< TIME *TIME*)
        (FERROR "Tried to schedule event in the past."))
  (ADD-EVENT-TO-HEAP
   (MAKE-EVENT TIME TIME
               FUNCTION EVENT-FUNCTION
               ARGUMENTS (COPYLIST EVENT-ARGUMENTS))))


;;; Everything starts up.

(DEFUN SIMULATION-STARTUP ()
  (LOOP FOR USER IN *USERS*
        DO
        (IF (EVENP (USER-ID USER))
            (START-TALKING USER)
            (EVENT #'CONTROL-PACKET-GENERATION
                   (+ *TIME* *CONTROL-PACKET-SPACING*)
                   USER))
        (EVENT #'USER-STATISTICS-COLLECTION
               (+ *TIME* *USER-STATISTICS-INTERVAL*)
               USER))
  (LOOP FOR LINK IN *LINKS*
        DO
        (EVENT #'UPDATE
               (+ *TIME*
                  (SI:RANDOM-IN-RANGE 0 *UPDATE-INTERVAL*))
               LINK)
        (EVENT #'LINK-STATISTICS-COLLECTION
               (+ *TIME* *LINK-STATISTICS-INTERVAL*)
               LINK)))

;;; A link begins passing the first packet in its queue to
;;; the next link in that packet's route.


(DEFUN PACKET-TRANSMISSION (LINK)
  (LET ((LAST-PACKET-SENT
         (LINK-PACKET-NOW-TRANSMITTING LINK)))
    (WHEN LAST-PACKET-SENT
      (INCF (LINK-NUMBER-OF-BITS-SENT LINK)
            (PACKET-LENGTH LAST-PACKET-SENT))
      (INCF (LINK-TOTAL-NUMBER-OF-BITS-SENT LINK)
            (PACKET-LENGTH LAST-PACKET-SENT))
      (IF (PACKET-LAST-IN-TALK-SPURT? LAST-PACKET-SENT)
          (REMOVE-USER-FROM-LINK
           (PACKET-SOURCE LAST-PACKET-SENT)
           LINK))))
  (LET ((PACKET (POP (LINK-QUEUE-FRONT LINK))))
    (SETF (LINK-PACKET-NOW-TRANSMITTING LINK) PACKET)
    (UNLESS (NULL PACKET)
      (INCF (LINK-NUMBER-OF-PACKETS-SENT LINK))
      (INCF (LINK-TOTAL-NUMBER-OF-PACKETS-SENT LINK))
      (DECF (LINK-QUEUE-LENGTH LINK))
      (LET ((PACKET-DELAY (- *TIME*
                             (PACKET-ARRIVAL-TIME PACKET))))
        (INCF (LINK-TOTAL-PACKET-DELAY LINK) PACKET-DELAY)
        (INCF (LINK-TOTAL-SQUARED-PACKET-DELAY LINK)
              (^ PACKET-DELAY 2))
        (INCF (AREF (LINK-PACKET-DELAY-HISTOGRAM LINK)
                    (MIN 10
                         (FIX (// PACKET-DELAY 0.002))))))
      (SETF (PACKET-FORWARD-CONTROL PACKET)
            (MIN (PACKET-FORWARD-CONTROL PACKET)
                 (LINK-CONTROL LINK)))
      (EVENT #'PACKET-TRANSMISSION
             (+ *TIME*
                (// (PACKET-LENGTH PACKET)
                    (FLOAT (LINK-CAPACITY LINK)))
                *PACKET-TRANSMISSION-DELAY*)
             LINK)
      (LET ((DESTINATION (POP (PACKET-ROUTE PACKET))))
        (IF (NULL DESTINATION)
            (EVENT #'PACKET-ABSORPTION
                   (+ *TIME*
                      (LINK-PROPAGATION-DELAY LINK)
                      *PACKET-ABSORPTION-DELAY*)
                   PACKET
                   (PACKET-DESTINATION PACKET))
            (EVENT #'PACKET-ARRIVAL
                   (+ *TIME*
                      (LINK-PROPAGATION-DELAY LINK)
                      *PACKET-ARRIVAL-DELAY*)
                   PACKET
                   DESTINATION)
            (EVENT #'PACKET-TAIL-ARRIVAL
                   (+ *TIME*
                      (// (PACKET-LENGTH PACKET)
                          (FLOAT (LINK-CAPACITY LINK)))
                      (LINK-PROPAGATION-DELAY LINK)
                      *PACKET-ARRIVAL-DELAY*)
                   PACKET
                   DESTINATION))))))

;;; A packet is received by its intended target user.

(DEFUN PACKET-ABSORPTION (PACKET USER)
  (LET ((DELAY (- (+ *TIME* *PACKET-ABSORPTION-DELAY*)
                  (PACKET-GENERATION-TIME PACKET))))
    (SELECTQ (PACKET-TYPE PACKET)
      (VOICE
       (INCF (USER-TOTAL-VOICE-PACKET-DELAY USER) DELAY)
       (INCF (USER-NUMBER-OF-VOICE-PACKETS USER))
       (SETF (USER-MAX-VOICE-PACKET-DELAY USER)
             (MAX (USER-MAX-VOICE-PACKET-DELAY USER)
                  DELAY)))
      (CONTROL
       (INCF (USER-TOTAL-CONTROL-PACKET-DELAY USER)
             DELAY)
       (INCF (USER-NUMBER-OF-CONTROL-PACKETS USER))
       (SETF (USER-MAX-CONTROL-PACKET-DELAY USER)
             (MAX (USER-MAX-CONTROL-PACKET-DELAY USER)
                  DELAY)))))
  (SETF (USER-RATE USER)
        (PACKET-FEEDBACK-CONTROL PACKET))
  (SETF (USER-PARTNER-RATE USER)
        (PACKET-FORWARD-CONTROL PACKET))
  (WHEN (PACKET-LAST-IN-TALK-SPURT? PACKET)
        (SETF (USER-TALKING? (USER-PARTNER USER)) NIL)
        (START-TALKING USER)))
```

```
(DEFUN REMOVE-USER-FROM-LINK (USER LINK)
  (SETF (LINK-USERS LINK)
        (DELQ (ASSQ USER (LINK-USERS LINK))
              (LINK-USERS LINK)))
  (DECF (LINK-NUMBER-OF-USERS LINK)))


;;; This is a separate function so it can be called at
;;; initialization time.

(DEFUN START-TALKING (USER)
  (SETF (USER-TALKING? USER) T)
  (EVENT #'VOICE-PACKET-GENERATION
         (+ *TIME*
            *PACKET-ABSORPTION-DELAY*
            (RANDOM-INTER-TALK-SPURT-SILENCE))
         USER))

;;; A packet begins to arrive at a link.

(DEFUN PACKET-ARRIVAL (PACKET LINK)
  (SETF (PACKET-ARRIVAL-TIME PACKET) *TIME*)
  ;; Enqueue packet.
  (LET ((OLD-QUEUE-FRONT (LINK-QUEUE-FRONT LINK)))
    (LET ((NEW-QUEUE-BACK (LIST PACKET)))
      (IF (NULL (LINK-QUEUE-FRONT LINK))
          (SETF (LINK-QUEUE-FRONT LINK) NEW-QUEUE-BACK)
          (RPLACD (LINK-QUEUE-BACK LINK)
                  NEW-QUEUE-BACK))
      (SETF (LINK-QUEUE-BACK LINK)
            NEW-QUEUE-BACK)
      (INCF (LINK-QUEUE-LENGTH LINK))
      (SETF (LINK-MAX-QUEUE-LENGTH LINK)
            (MAX (LINK-QUEUE-LENGTH LINK)
                 (LINK-MAX-QUEUE-LENGTH LINK))))
    ;; If link is idle, schedule instant transmission.
    (UNLESS (OR (LINK-PACKET-NOW-TRANSMITTING LINK)
                (NOT (NULL OLD-QUEUE-FRONT)))
      (EVENT #'PACKET-TRANSMISSION
             (+ *TIME* *PACKET-TRANSMISSION-DELAY*)
             LINK))))

;;; The tail end of a packet arrives at a link.

(DEFUN PACKET-TAIL-ARRIVAL (PACKET LINK)
  (WHEN (EQ (PACKET-TYPE PACKET) 'VOICE)
    (LET ((ACTIVE-USER (ASSQ (PACKET-SOURCE PACKET)
                             (LINK-USERS LINK))))
      (IF ACTIVE-USER
          (RPLACD ACTIVE-USER
                  (PACKET-SOURCE-RATE PACKET))
          (ADD-USER-TO-LINK
            (PACKET-SOURCE PACKET)
            LINK
            (PACKET-SOURCE-RATE PACKET))))))

(DEFUN ADD-USER-TO-LINK (USER LINK PACKET-LENGTH)
  (PUSH (CONS USER PACKET-LENGTH)
        (LINK-USERS LINK))
  (INCF (LINK-NUMBER-OF-USERS LINK)))

;;; A user creates a voice packet.

(DEFUN VOICE-PACKET-GENERATION (USER)
  (LET ((PACKET
         (MAKE-PACKET
          FEEDBACK-CONTROL (USER-PARTNER-RATE USER)
          LENGTH (FIX (// (USER-RATE USER) 50))
          ROUTE (USER-ROUTE USER)
          GENERATION-TIME *TIME*
          LAST-IN-TALK-SPURT?
          (IF *USERS-TALK-FOREVER*
              NIL
              (= 0
                 (RANDOM
                  *AVERAGE-TALK-SPURT-LENGTH*)))
          TYPE 'VOICE
          SOURCE USER
          SOURCE-RATE (USER-RATE USER)
          DESTINATION (USER-PARTNER USER))))
    (LET ((FIRST-LINK (POP (PACKET-ROUTE PACKET))))
      (EVENT #'PACKET-ARRIVAL
             (+ *TIME* *PACKET-GENERATION-DELAY*)
             PACKET
             FIRST-LINK)
      (EVENT #'PACKET-TAIL-ARRIVAL
             (+ *TIME* *PACKET-GENERATION-DELAY*)
             PACKET
             FIRST-LINK))
    (IF (PACKET-LAST-IN-TALK-SPURT? PACKET)
        (EVENT #'CONTROL-PACKET-GENERATION
               (+ *TIME* *CONTROL-PACKET-SPACING*)
               USER)
        (EVENT #'VOICE-PACKET-GENERATION
               (+ *TIME* (SI:RANDOM-IN-RANGE 0.018 0.022))
               USER))))

;;; A user creates a control packet.

(DEFUN CONTROL-PACKET-GENERATION (USER)
  (UNLESS (USER-TALKING? USER)
    (LET ((PACKET
            (MAKE-PACKET FEEDBACK-CONTROL
                         (USER-PARTNER-RATE USER)
                         LENGTH 10
                         ROUTE (USER-ROUTE USER)
                         GENERATION-TIME *TIME*
                         TYPE 'CONTROL
                         SOURCE USER
                         DESTINATION (USER-PARTNER USER)))
      (EVENT #'PACKET-ARRIVAL
             (+ *TIME* *PACKET-GENERATION-DELAY*)
             PACKET
             (POP (PACKET-ROUTE PACKET)))
      (EVENT #'CONTROL-PACKET-GENERATION
             (+ *TIME* *CONTROL-PACKET-SPACING*)
             USER))))

;;; A link updates its control.

(DEFUN UPDATE (LINK)
  (MULTIPLE-VALUE-BIND (MAX-RATE FLOW)
      (MAXIMIZE-AND-SUM-RATES-ON LINK)
    (IF (OR (SELECTQ *UPDATE-PROTOCOL*
              (MOSELY (NOT ( MAX-RATE
                               (LINK-CONTROL LINK))))
              (HAYDEN NIL))
            (ZEROP (LINK-NUMBER-OF-USERS LINK)))
        (EVENT #'UPDATE
               (+ *TIME* *UPDATE-ATTEMPT-INTERVAL*)
               LINK)
        (SETF (LINK-CONTROL LINK)
              (FUNCALL *UPDATE-FUNCTION*
                       LINK MAX-RATE FLOW))
        (EVENT #'UPDATE
               (+ *TIME* *UPDATE-INTERVAL*)
               LINK))))

(DEFUN MAXIMIZE-AND-SUM-RATES-ON (LINK)
  (LOOP FOR (USER . RATE) IN (LINK-USERS LINK)
        MAXIMIZE RATE INTO MAX-RATE
        SUMMING RATE INTO SUM
        FINALLY
        (RETURN MAX-RATE SUM)))

(DEFUN MOSELY-UPDATE-FUNCTION (LINK MAX-RATE FLOW)
  (LET ((ALPHA 1.0)
        (EFFECTIVE-CAPACITY (* 0.8 (LINK-CAPACITY LINK))))
    (MAX (MIN (+ MAX-RATE
                 (// (* ALPHA (- EFFECTIVE-CAPACITY FLOW))
                     (LINK-NUMBER-OF-USERS LINK)))
              EFFECTIVE-CAPACITY)
         (// EFFECTIVE-CAPACITY
             (LINK-NUMBER-OF-USERS LINK)))))

(DEFUN HAYDEN-UPDATE-FUNCTION (LINK MAX-RATE FLOW)
  MAX-RATE
  (LET ((ALPHA 1.0)
        (EFFECTIVE-CAPACITY (* 0.8 (LINK-CAPACITY LINK))))
    (MAX (MIN (+ (LINK-CONTROL LINK)
                 (// (* ALPHA (- EFFECTIVE-CAPACITY FLOW))
                     (LINK-NUMBER-OF-USERS LINK)))
              EFFECTIVE-CAPACITY)
         (// EFFECTIVE-CAPACITY
             (LINK-NUMBER-OF-USERS LINK)))))

(DEFUN JAFFE-UPDATE-FUNCTION (LINK MAX-RATE FLOW)
  MAX-RATE
  (LET ((ALPHA 1.0)
        (EFFECTIVE-CAPACITY (* 0.8 (LINK-CAPACITY LINK)))
        (CONTROL (LINK-CONTROL LINK)))
    (MAX (MIN (+ CONTROL
                 (// (* ALPHA (- EFFECTIVE-CAPACITY
                                 FLOW
                                 CONTROL))
                     (+ (LINK-NUMBER-OF-USERS LINK) 1)))
              EFFECTIVE-CAPACITY)
         (// EFFECTIVE-CAPACITY
             (+ (LINK-NUMBER-OF-USERS LINK) 1)))))

;;; Some network behavior statistics are recorded.

(DEFUN LINK-STATISTICS-COLLECTION (LINK)
  (COND ((LINK-PRINT-STATISTICS? LINK)
         (FORMAT *LINK-STAT-STREAM*
                 " % 10D 2,1,10$ 10D 10D 10D 10D 10D 10D"
                 (LINK-ID LINK)
                 *TIME*
                 (LINK-NUMBER-OF-BITS-SENT LINK)
                 (FIXR (LINK-CONTROL LINK))
                 (LINK-NUMBER-OF-USERS LINK)
                 (LINK-MAX-QUEUE-LENGTH LINK)
                 (LINK-QUEUE-LENGTH LINK)
                 (LINK-NUMBER-OF-PACKETS-SENT LINK))
         (ALTER-LINK LINK
                     NUMBER-OF-BITS-SENT 0
                     MAX-QUEUE-LENGTH 0
                     NUMBER-OF-PACKETS-SENT 0)))
  (EVENT #'LINK-STATISTICS-COLLECTION
         (+ *TIME* *LINK-STATISTICS-INTERVAL*)
         LINK))

(DEFUN USER-STATISTICS-COLLECTION  (USER)
  (COND
   ((USER-PRINT-STATISTICS? USER)
    (FORMAT
     *USER-STAT-STREAM*
     " % 10D 2,1,10$ 2,1,10$ 2,1,10$ 2,1,10$ 2,1,10$ 10D 10D"
     (USER-ID USER)
     *TIME*
     (SAFE-// (USER-TOTAL-VOICE-PACKET-DELAY USER)
              (USER-NUMBER-OF-VOICE-PACKETS USER))
     (USER-MAX-VOICE-PACKET-DELAY USER)
     (SAFE-// (USER-TOTAL-CONTROL-PACKET-DELAY USER)
              (USER-NUMBER-OF-CONTROL-PACKETS USER))
     (USER-MAX-CONTROL-PACKET-DELAY USER)
     (FIX (USER-RATE USER))
     (FIX (USER-PARTNER-RATE USER)))
    (ALTER-USER USER
                TOTAL-VOICE-PACKET-DELAY 0
                NUMBER-OF-VOICE-PACKETS 0
                MAX-VOICE-PACKET-DELAY 0
                TOTAL-CONTROL-PACKET-DELAY 0
                NUMBER-OF-CONTROL-PACKETS 0
                MAX-CONTROL-PACKET-DELAY 0)))
  (EVENT #'USER-STATISTICS-COLLECTION
         (+ *TIME* *USER-STATISTICS-INTERVAL*)
         USER))


;;; ********************************************************
;;;                   Random stuff.
;;; ********************************************************

(DEFUN RANDOM-INTER-TALK-SPURT-SILENCE ()
  0.0001)

(DEFUN PRINT-LINK-HISTOGRAMS ()
  (FORMAT *LINK-STAT-STREAM* "
                        Av. squared
Link Packets  Av. wait  wait      Av. flow ")
  (LOOP FOR LINK IN *LINKS*
        DO
        (FORMAT *LINK-STAT-STREAM*
                " % 4D 8D 6,1,10$ 6,1,10$ 2,1,10$"
                (LINK-ID LINK)
                (LINK-TOTAL-NUMBER-OF-PACKETS-SENT LINK)
                (// (LINK-TOTAL-PACKET-DELAY LINK)
                    (LINK-TOTAL-NUMBER-OF-PACKETS-SENT LINK))
                (// (LINK-TOTAL-SQUARED-PACKET-DELAY LINK)
                    (LINK-TOTAL-NUMBER-OF-PACKETS-SENT LINK))
                (// (LINK-TOTAL-NUMBER-OF-BITS-SENT LINK)
                    *TIME*))
        (LOOP FOR I FROM 0 TO 10
              DO
              (FORMAT
               *LINK-STAT-STREAM* " 2,1,8$"
               (// (* 100.0
                      (AREF (LINK-PACKET-DELAY-HISTOGRAM LINK)
                            I))
                   (LINK-TOTAL-NUMBER-OF-PACKETS-SENT LINK))))
        ))
```

```
(DEFUN SAFE-// (X Y)
  (IF (ZEROP Y)
      0
      (// X Y)))
```

Biographical Note

The author was born in Pittsburgh, PA, May 16, 1953. At the time, her parents resided in the nearby suburb, Mars. For years she took great delight in being able to tell people, quite truthfully, that she was from Mars. In 1970, she graduated from the University of Illinois High School in Urbana, where she learned to love mathematics. In 1974, she received a B.A. in Mathematics from the University of Illinois. After a year working, she found out what the degree was worth, and went back to school. In 1977, she received a B.S. in Electrical Engineering, also from the University of Illinois. In 1979, she received an S.M. in Electrical Engineering from the Massachusetts Institute of Technology, and in 1980, the degree of Electrical Engineer. The Ph.D. will be her fifth and last degree, unless she decides to go to law school.

In additional to her professional interests, the author is fascinated by rotational motion, and enjoys bicycling, wood turning, and hand spinning. She also collects and restores antique spinning wheels, and has invented many designs for paper-folded polyhedra.

References

[1] B. Gold, "Digital Speech Networks", Proc. IEEE, Vol. 65, Dec. 1977.

[2] H. Frank and I. Gitman, "Economic Analysis of Integrated Voice and Data Networks; A Case Study", Proc. IEEE, Vol. 66, Nov. 1978.

[3] K. Bullington and J. M. Fraser, "Engineering Aspects of TASI", Bell System Technical Journal, Vol. 38, Mar 1959.

[4] S. J. Campenella, "Digital Speech Interpolation", COMSAT Technical Review, Vol. 6, Spring 1976.

[5] J. A. Sciulli and S. J. Campanella, "A Speech Predictive Encoding Communication System for Multichannel Telephony", IEEE Trans. Comm., Vol. COM-21, July 1973.

[6] M. Gerla and L. Kleinrock, "Flow Control: A Comparative Survey", IEEE Trans. Comm., Vol. COM-28, April 1980.

[7] G. Karog, L. Fransen and E. Kline, "Multirate Processor (MRP)", Naval Research Laboratory Report, Sept. 1980.

[8] T. Bially, B. Gold, and S. Seneff, "A Technique for Adaptive Voice Flow Control in Integrated Packet Networks", IEEE Trans. Comm., Vol. Com-28, March 1980.

[9] H. Hayden, "Voice Flow Control in Integrated Packet Networks", M. S. Thesis, Dept. of Elec. Eng. and Comp. Science, Mass. Inst. of Technology, Cambridge, MA, 1981.

[10] J. M. Jaffe, "A Decentralized 'Optimal' Multiple-User Flow Control Algorithm", ICCC .1980 Conference Record.

[11] E. M. Gafni, "The Integration of Routing and Flow Control for Voice and Data in a Computer Communication Network", Ph. D. Dissertation, Dept. of Elec. Eng. and Comp. Science, Mass. Inst. of Technology, Cambridge, MA, Aug. 1982.

[12] E. Gafni and D. Bertsekas, "Dynamic Control of Session Input Rates in Communication Networks", LIDS Report.

[13] D. P. Bertsekas, J. N. Tsitsiklis, M. Athans, "Convergence Theories of Distributed Iterative Process: A Survey", M.I.T. LIDS Report P-1342, Dec. 1983

[14] D. P. Bertsekas, "Asynchronous Computation of Fixed Points", Mathematical Programming, Vol. 27 (1983) pp. 107-120.

[15] P. T. Brady, "A Statistical Analysis of On-Off Patterns in 16 Conversations," Bell System Technical Journal, Vol. 47,Jan. 1968, pp. 73-91.