Research Experiences for Undergraduates (REU) Program: Project Summary

Darren H. Mutz   Ian K. Struble

March 21, 1997

DRAFT

1 Introduction

Broadband integrated services digital networks (B-ISDN) will be used for the transmission of digital information from a variety of different sources simultaneously (e.g., video, audio, facsimile data). The asynchronous transfer mode (ATM) protocol reduces these data streams to ATM-cells that are multiplexed and conveyed over the network to their destination.

Previous work in traffic measurement models under support of NSF Grant NCR-9103998 has attempted to reconstruct the component rates at which different users are delivering ATM-cells to the network. The motivation for such a model is that any traffic control policy would need to at least be capable of determining the fraction of network load each sender is responsible for. The devised scheme models the hidden process of switching between traffic conveyed by one user to another as a Markov chain and reconstructs the component sending frequencies given the observed inter-arrival times of ATM-cells.

Because of the technical difficulty of measuring ATM network traffic at the cell level, our work has focused on verifying an abstraction of the original problem the model was designed for, that of reconstructing the constituent sender frequencies by recording the inter-arrival times of messages arrive at the receiver. We would argue that, for relatively small message sizes (one thousand bytes) and for large inter-send delays (one thousand microseconds), the flow of these messages closely approximates the behavior of the flow of individual ATM-cells.

2 Our experiments

2.1 Architecture of the test platform

To make our measurements, we used a cluster of Silicon Graphics Indy workstations connected to each other and to a Silicon Graphics Challenge system via a dedicated fiber optic ATM network. The Challenge is of significantly more computational power than the workstations with two MIPS R4400 processors clocked at 150 MHz; the workstations employ a single MIPS R4600 clocked at 132 MHz. The workstations and the Challenge both use Fore Systems ATM network interfaces. The network is configured in a star topology with switching provided by a Fore Systems ASX-200. Both the Challenge system and the Indy workstations run version 5.3 of the IRIX operating system.

At a high level, our software architecture consists of a group of user level sending applications that are run one to a workstation, with an equal number of user level receiving applications running simultaneously on the Challenge system. This approach was taken in order to ensure that all traffic was routed through a single receiving network interface.

The traffic is generated synthetically, with senders delaying for a fixed or stochastically varied amount of time. The communications channel is established and controlled by the application programmer's interface provided by Fore Systems.

The time delay of the sending applications as well as the time stamping of incoming messages by the receiver is achieved using a function call provided by the standard library. Initial plans involved using the high resolution timer provided by the IRIX kernel, but use of this timer is restricted to kernel modules. Kernel modules are notoriously difficult to implement and debug, so a user level application was settled upon.

2.2 Measurements

Each of the running processes that are configured to receive messages (one or more grouped ATM cells) over the network timestamp the arrival of each such message. After the run has completed, these sorted lists of arrival times are merged and then a subtraction is performed between adjacent timestamps to yield the set of inter-arrival times.

¿From this set, one can construct a distribution plot of the observed inter-arrival times at the receiver. In the case of senders at a constant rate, it is generally trivial to identify the component rates from this plot alone. However, when the inter-send delays are very close or when they are stochastically varied, it becomes very difficult to reliably identify the different components.

In this case we enlist the help of the model and compute the U_n values according to the following recurrence:

$$U_n = (1-\theta)U_{n-1} + \theta T_n$$

Where T_n is the nth observed inter-arrival time. Finally, the range of U_n values observed is discretized and a histogram is plotted. A sender's rate will appear as the center of a concentration of data points when viewed in this manner. In this way, we are able to recover the constituent rates of transmission..

2.2.1 Two senders, constant rate

Here we demonstrate the functionality of the model for the case of two senders sending at a very similar, but constant rate. One sender is sending messages with a delay of 0.001 seconds while the other sends with a delay of 0.0015 seconds. Each sends 1000 messages to the receiver. The inter-send delay times here are distinct, but it is still difficult to identify the component rates of transmission by simply looking at the distribution of the observed inter-arrival times, shown in Figure 1.

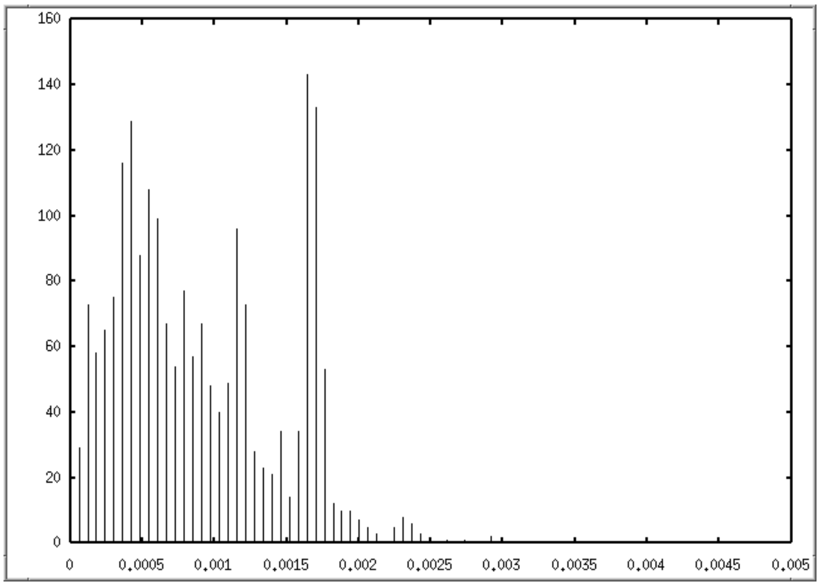

Figure 1: Distribution of observed inter-arrival times (constant rate).

When we apply the model however, we see the following distribution in Figure 2.

The process sending at 0.0015 seconds is clearly visible here. The location of the 0.001 second rate is harder to pinpoint, but lies near 0.001 seconds.

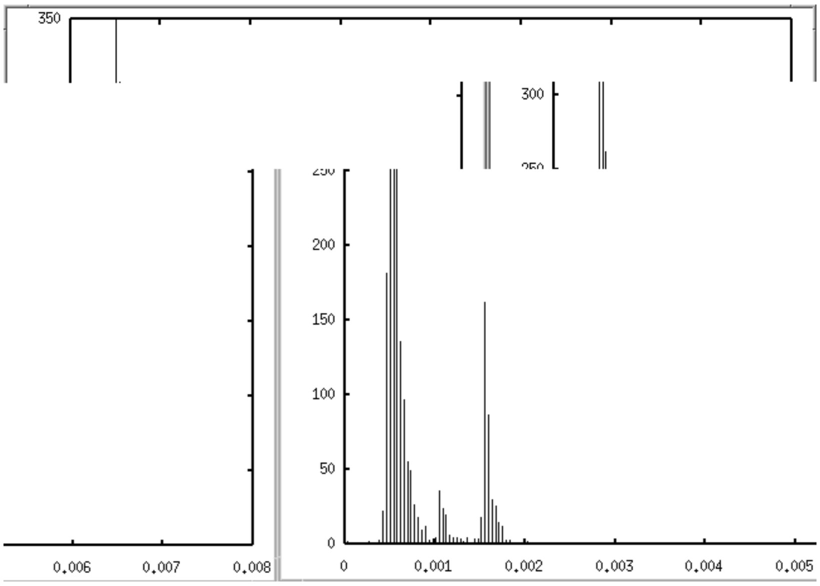

Figure 2: Distribution of computed U_n values (constant rate).

2.2.2 Two senders, varied rate.

In this example we allow the sending processes to vary about their mean values, which are the same as in the previous example. The intuition here is that, in a real-world application of the model, sending processes are likely to be influenced by random factors on their respective systems (e.g., high system load or contention for the network interface).

It is important to note that, with sending rates that have similar means and are allowed to vary in this manner, it is very difficult to pick out the individual rates of transmission by observing the inter-arrival times alone (Figure 3). Applying the model to the data however, we are able to identify two sending processes near 0.001 and 0.0015 seconds (Figure 4).

This data, while being significantly more noisy that our previous observation, indicates two sending processes near 0.001 and 0.0015 seconds.

3 Problems with our method.

Our results do not currently reflect the the level of accuracy required to be a of use in a real-world application. Our observations do not indicate very precisely the rate at which a particular sender is delivering data to the network, and the fraction of traffic each represents (equal in this case) is not accurately conveyed.

In both examples, the distribution closer to zero appeared larger and more

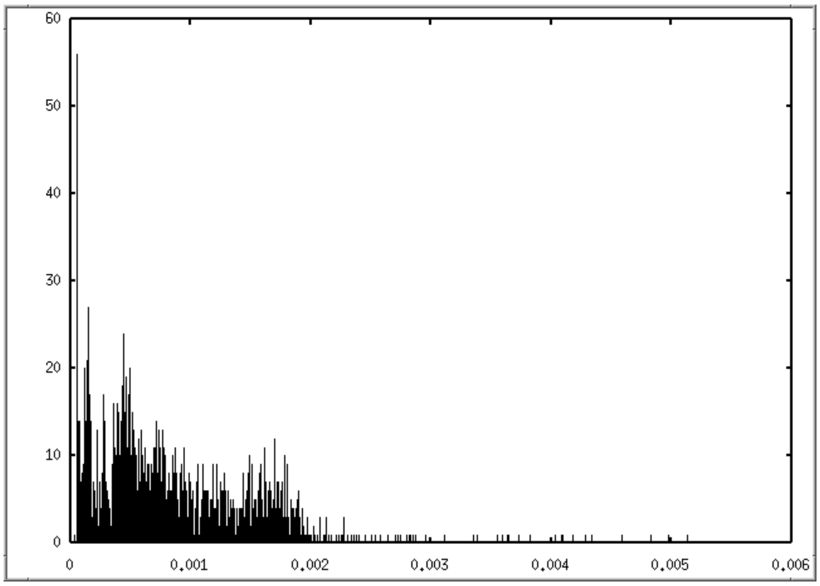

Figure 3: Distribution of observed inter-arrival times (varied rate).

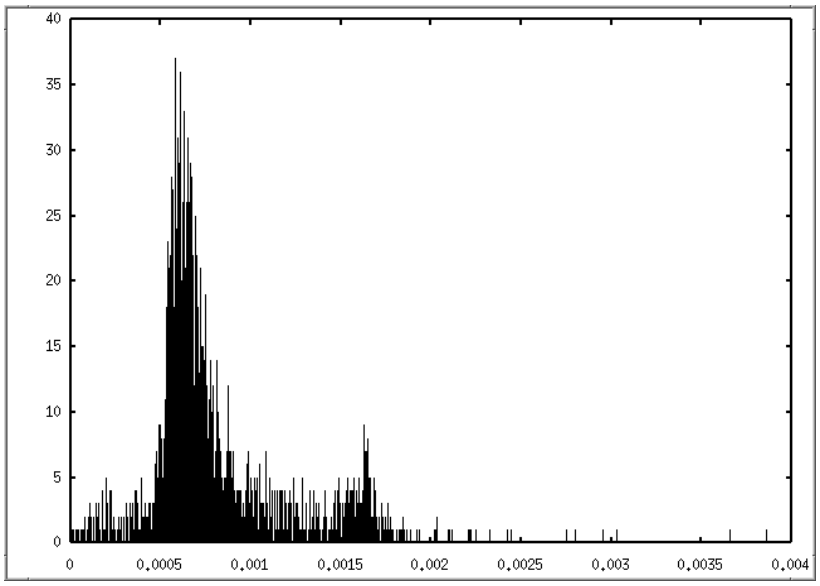

Figure 4: Distribution of calculated U_n values (varied rate).

skewed in the zero direction than the second distribution. It is our feeling that this is caused by buffering that occurs at different levels in the software and hardware interface. Assuming a sizable inter-send delay, buffering a large group of messages will have the effect of the first message being received late, but the following messages arriving in rapid succession. This would cause the observed inter-arrival times to be skewed toward zero and likewise cause the U_n values to drift toward zero as we observed.

4 Conclusion

To take this project to the next level, we need to identify and correct for the effects that are skewing our results or refine the model we are using. We would also like to be able to generalize our method to be able to identify any number of senders.

One way of eliminating part of the distorting effects would be to write a kernel module to send and receive messages on the network. This would greatly increase the resolution of the timing measurements as well as eliminate almost all buffering effects in the software domain. One problem with this approach is that we would like to eventually be able to apply this model to standard network monitoring applications. These applications typically run at the user level and so would be susceptible to the same types of effects.

To generalize to an arbitrary number of senders, we would rely only on the claims of the original model and the computational abilities of the receiving system; there is the possiblility that a large number of receiving process running simultaneously on the receiving platform would significantly reduce the accuracy of the timing measurements. However, if we coupled this apprach with a kernel module on the receiving side, we would run only one such receiving process which would keep track of all incoming messages, regardless of the sender.