An Empirical Evaluation of Virtual Circuit Holding Times in IP-over-ATM Networks

H. Saran and S. Keshav[1]

Department of Computer Science and Engineering
Indian Institute of Technology
Hauz Khas, New Delhi 110016, India

1

Abstract

When carrying Internet Protocol (IP) traffic over an Asynchronous Transfer Mode (ATM) network, the ATM adaptation layer must determine how long to hold a virtual circuit opened to carry an IP datagram. We present a formal statement of the problem and an empirical study of holding time policies taking network pricing into account. We find that IP traffic shows temporal locality of reference and so Least Recently Used (LRU)-based policies perform well. A systemwide timeout, which is a special case of an LRU policy, is quite effective when the timeout value is chosen correctly. The policies we propose are easy to implement and solve the problem satisfactorily.

1 Introduction

The TCP/IP based Internet is the world's largest computer network with more than a million interconnected computers. This network is based on the connectionless Internet Protocol (IP). However, future networks will, at least at the lowermost layers, be connection-oriented, with data-link layer connectivity being provided by Asynchronous Transfer Mode (ATM) style networks. Thus, to protect existing investments, it is necessary to devise mechanisms to carry IP traffic on ATM networks.

We assume that IP traffic will be sent over ATM circuits by means of dual-ported routers (such as for a LAN-interconnect application). In this scenario, the router opens and closes virtual circuits (VCs) on behalf of arriving IP datagrams. The arrival of an IP datagram should cause a circuit to be established, but it is not clear how long an open circuit should be held (a problem that was faced while carrying IP over X.25 as well [1]). While one could open and close a circuit for each packet, it is likely that the connection setup time will be large, and so it would be desirable to amortize the connection setup time over many packets. Thus, the ATM adaptation layer would need to determine that it is no longer useful to hold a circuit and close it. This decision must be made heuristically, since IP datagrams do not carry any information about how long a higher layer conversation, if any, lasts.

In this paper, we formulate the holding time problem and use an empirical approach to model IP traffic. We show that an LRU-based approach allows efficient holding time decisions. The next section presents background and previous work. Section 3 is devoted to workload modeling. Sections 4 and 5 study the performance of different holding time policies assuming one of two pricing schemes. Finally, Section 6 presents our conclusions.

2 Background and Previous Work

In this section, we describe the issues in designing a VC holding policy, our assumptions, and previous work.

2.1 Background

Four factors determine how long a VC should be held:

1. the pricing structure in the network
2. the user's loss of utility from a unit increase in delay

[1]On leave from AT&T Bell Laboratories, Murray Hill, NJ 07974.

3. the traffic arrival pattern in the network

4. the maximum number of open VCs allowed per end-system

The pricing structure strongly influences the holding time policy. Users are typically charged for opening a connection, a per-packet usage charge, and sometimes an additional holding time charge. The cost of opening a circuit itself has two components: a monetary cost charged by carriers to pay for call setup, and a 'delay' cost. The delay cost measures the loss of utility to a user because of the delay in opening a circuit. A system manager can vary this cost to choose the tradeoff between setup delays and holding time costs for a particular topology and traffic workload.

The traffic arrival stream also influences the holding policy. For example, if the traffic arrives in a bursty fashion, with long inter-burst intervals, then the router should close the connection at the end of each burst. Unfortunately, the traffic arrival pattern depends on user behavior and is an unknown quantity.

Some connection-oriented networks have an upper limit on the number of VCs that any single system may open. For example, in an X.25 network, this is typically in the range 32 to 128 [1]. This affects the holding time policy, since the router must close an open circuit if a datagram arrives when there are no circuits available.

In this paper, we assume that

1. The cost of delay can be quantified by the system manager. However, we find that even if this quantification is inexact, LRU-based policies (to be discussed later) perform well.

2. The data we collected is representative of traffic in typical IP networks.

3. The measured LAN traffic will actually be carried over a WAN. This assumption may seem surprising at first, since current LAN and WAN traffic characteristics differ widely. However, we anticipate that as high speed ATM WANs become available, higher throughputs and lower delays will make WAN and LAN traffic look similar. For example, given high speed wide area networks, it is feasible to mount remote file systems, and open client server applications (such as X) over a WAN, whereas these options are unviable today.

4. The VC opened by a router has sufficient capacity to carry arriving IP traffic.

2.2 Holding time policies

The *optimal* holding policy is one that is non-causal, that is, it knows about future packet arrivals. While the optimal policy is unachievable, it provides a benchmark against which to compare all other policies.

A natural class of causal (achievable) policies are those that emulate the optimal policy by predicting packet arrivals. Let the history $H_i(t)$ of conversation i be the sequence of packet arrival times for conversation i till time t. A *predictor* $\phi(H(t), t)$ is a function that takes as input the history of a conversation and the current time t, and outputs an estimated next arrival time for a packet on that conversation. A *predictor based policy* uses a predictor to estimate the future arrivals from each conversation and make its decision in the same way as the corresponding optimal policy. Thus, the essence of the problem is to design good predictors.

One subclass of predictor based policies is Timer-based, where each open VC is associated with a timer. The timer is set using some algorithm, and when the timeout happens, the VC is marked eligible to be closed. Depending on the pricing policy, the VC is either closed immediately or closed when there is a demand for another VC. We introduce another subclass of holding time policies, LRU-based policies, in Section 3.2.

The ATM adaptation layer may offer host-to-host connectivity either by using one connection to send all the IP traffic between a pair of IP addresses or by using different connections for different port numbers on the same IP host. We investigated both situations and find that our results, in terms of the relative performance of the different policies studied, apply to both situations. For reasons of space, we only present results for VCs between IP pairs.

We only consider policies that do not assume any correlation between different conversations. Since many IP conversations are duplex, some policies may exploit the correlation between the two simplex halves of a duplex conversation. However, the ATM adaptation layer at a router would probably not have the enough information to pair the respective connections.

2.3 Previous Work

The holding time problem arises naturally in carrying connectionless protocols such as IP over Virtual Circuit oriented networks such as X.25 and Datakit. While existing implementations embody several holding time policies, we are not aware of a formal statement of the holding time problem or a comparative study of these policies in the literature.

The use of LRU policies is widespread in computer systems, particularly for paging memory (a similar problem). While LRU-based holding times have been used in the past [1], they have not been studied in detail, particularly with reference to specific pricing policies.

3 Workload modeling

We collected several traces of packet arrivals on a loaded Ethernet by using the Unix 'etherfind' command. This command places the Ethernet interface in promiscuous mode and collects all the Ethernet headers received at the board, along with a time-stamp. This command was run on Ethernets at the Indian Institute of Technology, Delhi, University of Southern California, Los Angeles, AT&T Bell Laboratories, Murray Hill, and University of California, Berkeley. We collected several traces of 2000 - 5000 consecutive packets at each network.

The four environments had quite distinct characteristics: the data from UCB and USC was taken from LANs with a large number of workstations which were all active, and there were many simultaneously active conversations. The dataset at IIT Delhi was taken from a LAN that had a few workstations and a number of PC's using TCP/IP. The number of active connections here was significantly lower and the data consisted of a fewer number of connections being sampled for a larger period of time. The AT&T Bell labs dataset was taken on a network with a small number of active workstations and had somewhat similar characteristics to the IIT Delhi dataset. All data was filtered to remove broadcast packets.

3.1 Data Analysis

Even before we collected traces, based on the application level characterization work by Caceres et al [2], our intuition was that packet arrivals on each conversation would alternate between two time-scales that we dubbed 'user time-scale' and 'network time-scale'. The idea is that some usage of the network must be mediated by a human user, and thus shows relatively long inter-packet gaps, while other usage is mediated directly by a computer, and so will have relatively shorter interpacket gaps. As an example, during an FTP session, a human user may type 'get $< filename >$', where each keystroke is at the user time-scale. However, in an uncongested network, the

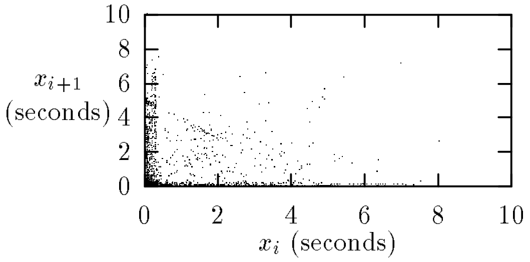

Figure 1: Scatter plot

response would be a stream of back-to-back packets at the network time-scale. A user would typically not generate packets at that speed. The presence of two time-scales would result in temporal locality of reference, since packet arrivals at the the network time scale would all refer to the same conversation.

We tested this hypothesis in two ways: by looking at scatter plots, and by examining the frequency of reference to each level of an LRU stack.

Let the sequence of inter-packet intervals be $[x_i]$. A scatter plot graphs (x_i, x_{i+1}) for each value of i. A typical scatter plot is shown in Figure 1. Note that while most points are close to the origin, there is some clustering along the axes. This indicates that the most common behavior is that of successive short inter-packet gaps (drawn from the network time scale), with longer gaps once in a while (drawn from the user time scale). Two long gaps occur together rarely. Since the density along the axes is nearly uniform, the user level interpacket arrival time distribution can be approximately modeled by uniform random distribution.

More insight was gained by building a simulator that read a trace, and pulled each reference to a conversation to the top of a stack. It also kept track of the number of references to each level of the stack. A little thought shows that if our hypothesis is true, then the frequency of references to the upper levels of the stack would be higher than the frequency of references to the lower levels of the stack. Indeed, all our data show a steep decline in the frequency of reference to a stack level as the depth increases (Figure 2 shows two examples). This confirms the hypothesis and also indicates strong temporal locality of reference.

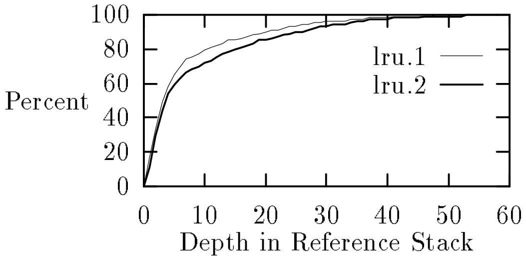

Figure 2: Cumulative references vs LRU stack depth

3.2 LRU-based policies

The presence of temporal locality in the data suggests that if a conversation has been pushed to a lower level in the LRU stack, then it is less likely to be referenced soon. Thus, if we drop conversations that have been pushed beyond some threshold level of the stack, it is likely that we will be dropping the right ones. This is the key idea in an LRU-based holding time policy.

Note that LRU can be viewed as a predictor based scheme where the prediction of future arrival on a VC is a monotonically increasing function f of the elapsed time since last packet arrival on that VC. We may choose any monotone function to work with, the simplest being the identity function. Note that this predictor uses only the last packet arrival time information, and ignores the rest of the history.

In the rest of the paper, we will present two reasonable pricing schemes, and show how a LRU based approach helps us in selecting the right conversation to drop in the context of each pricing scheme. The point to remember is that the presence of temporal locality in the data is what makes LRU attractive.

4 Pricing scheme 1: Paging model

In this pricing model, a fixed amount is charged for the total number of connections provided to a site. There may be an additional per-packet charge or a call setup charge. We will assume that there is no holding time charge, so that an open VC should be held until the limit on the number of open connections has been reached [2]. This type of pricing scheme is common in X.25 networks.

[2] We discuss the case where there is a holding time charge in the next section.

4.1 Holding policies

In this model, the holding time problem reduces to a paging problem, where a fault occurs when a packet arrives at a VC that is currently not open [1]. On a fault, an existing VC must be closed and a new VC opened. Thus, the holding time policy is simply an algorithm to choose the VC which is to be closed. A predictor based policy would use a predictor to estimate the future arrival times for each VC and then discard the conversation with the largest predicted idle period. We now discuss four classes of holding policies, then evaluate them on our data sets.

The first policy, the optimal policy, is to simply drop the conversation which will be inactive for the longest period of time. Since the optimal policy requires knowledge of the future, it is not implementable. To put this in perspective, we studied a second policy that uses no knowledge about the conversations, and discards an existing conversation at random when a new one needs to be opened.

The third class of policies was timer based. The strategy was to estimate the distribution of interpacket gaps for a particular VC based on past history. The estimation algorithm was derived from Jacobson's work on good estimators for round trip times [3]. Let I_k be the kth inter-reference gap for a VC. The mean inter-reference gap $\hat{I}$ and the standard error $\hat{\sigma}_k$ were estimated as:

$$\hat{I}_{k+1} = \alpha I_k + (1-\alpha)\hat{I}_k, 0 < \alpha < 1$$

$$\sigma_k = |\hat{I}_k - I_k|$$

$$\hat{\sigma}_{k+1} = \alpha\sigma_k + (1-\alpha)\hat{\sigma}_k$$

After each packet arrival on a VC, a timer is set to

$$\text{timeout}_{k+1} = \hat{I}_{k+1} + 2\hat{\sigma}_{k+1}$$

If no reference has occurred to a VC before its timeout, the VC is marked eligible to be dropped. On a fault, we drop the eligible VC with the most elapsed time since last packet arrival. In the case there are no eligible VCs, we drop the VC with the largest remaining timer value.

We studied a number of variations of this basic scheme. For example, note that the timeout for a VC is set at time t_k when the last packet arrives at that VC. However, the decision point t_j where we must choose to drop or hold the VC, comes later. At this time, the quantity we actually need is X_{t_j}, the expected time of next reference to this VC given that no packets arrived in the interval $[t_k, t_j]$. This depends upon the distribution for I_k and the elapsed

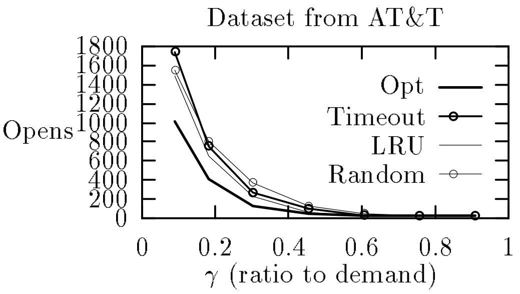

Figure 3: Total cost vs γ: Paging model

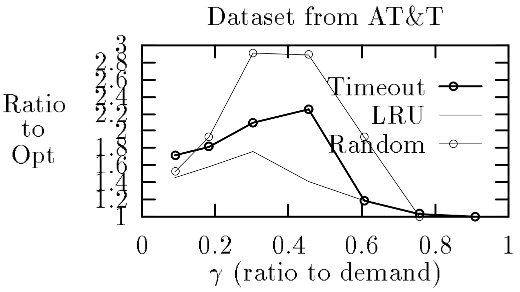

Figure 4: Normalized cost vs γ: Paging model

time $t_j - t_k$. Using several reasonable models for the nature of this distribution we derived formulas which approximated X_{t_j}. However, we found that none of the variations, despite the increase in complexity, had any significant effect on the performance of timeout based schemes. Hence, for reasons of space, we will not describe these variations further. In recent work, Lund, Phillips and Reingold have shown that maintaining an explicit histogram for I_k can improve prediction accuracy [4]. However, this requires substantial additional storage at each router.

The fourth policy was LRU-based and simply closed the VC at the bottom of the LRU stack when a new VC is needed.

4.2 Evaluation of Policies

All four holding time policies were simulated on each of the traces. The input parameters to the simulator are the trace file, the limit on the number of available VCs (denoted γ) and the holding policy to use. Each trace contains the packet arrival time, source, destination and port number for each packet. The statistics required by individual holding time policies were maintained and updated when each packet was processed. To implement the non-causal policy, we pre-processed the data to obtain information about the next arrival time.

Each trace has a certain total number of connections on it, denoted *demand*. Thus if the demand is less than γ, a connection is never dropped. In our study we varied γ from 10% of the demand to 90% of the demand and measured the number of connection establishments (opens). To be able to compare across datasets, we measure γ in terms of its ratio to the demand. Figure 3 shows the results for a representative trace. As expected, the number of opens decreases as γ is increased. The non-causal strategy is significantly better than any of the others though the gap seems to narrow as γ increases. This agrees with expectations.

To get a better feel of the relative performance of the strategies, we normalized the cost to the cost of the optimal strategy. Figure 4 shows a representative result. All the schemes performed well when running at 90% demand, when the ratio to optimum was close to one. We notice that LRU is consistently better than the others though the difference narrows when γ is larger than roughly 0.6.

Table 1 shows the normalized cost of a policy averaged over all the traces from all sites. Surprisingly, Timeout is only slightly better than Random (about 4%). LRU performs better than either of these policies. As γ is increased from 10% to 90% of demand, we see that the performance of LRU improves steadily whereas that of Random and Timeout first degrades and then improves.

The poor performance of Timeout was surprising. We investigated to check whether our prediction strategy was doing a reasonable job. We found that $62-69\%$ of the inter-arrival gaps lie in the interval $[\hat{I}-\hat{\sigma}, \hat{I}+\hat{\sigma}]$ and $78-87\%$ of the inter-arrival gaps lie in the interval $[\hat{I}-2\hat{\sigma}, \hat{I}+2\hat{\sigma}]$. So, the packet-arrival pattern does fit the model assumed by Timeout. The problem is that while the network time scale interarrival time distribution is captured by Timeout, the decision to page out a conversation depends on detecting relatively large periods of inactivity which is based on the tail of the inter-arrival time distribution corresponding to the user time scale. In terms of our observations in Section 3, in a cluster of closely arriving packets, Timeout predicts successive arrivals well, but when the large gap occurs, Timeout does poorly, since it has tuned its parameters to the preceding burst. Even a small gap after a burst causes a timeout. On the other hand LRU does well, since bursty VCs stay on top of the LRU stack and long

Table 1: Relative performance: Paging model

γ	Timeout	LRU	Random
0.09	1.98	1.59	1.96
0.18	2.10	1.55	2.19
0.30	2.30	1.51	2.37
0.45	1.93	1.38	2.25
0.61	1.33	1.25	1.57
0.76	1.14	1.13	1.15
0.91	1.05	1.05	1.06

gaps are compared on an absolute time scale.

Recall that the LRU scheme's predictor uses only the last packet arrival point in the history, and ignores the rest. This should contrasted with the more complex prediction methods used by the timer based schemes, which, in a sense, use the entire past history to construct a predictor. One may think of many intermediate solutions. For instance we investigated policies that use a little more history than LRU– for example, policies that use the last two arrival points in the history. We were not able to get any consistent improvements over LRU using these variations.

5 Pricing scheme 2: Holding cost model

Future ATM networks should allow each end-system to open a large number of virtual circuits, so we do not expect the number of open VCs to be the resource constraint. However, we expect there to be a charge for each unit of time that a circuit is kept open, whether or not it is used, so as to encourage users to free reserved per-VC resources when not needed. This holding time charge fundamentally changes the way in which the holding time policy needs to work.

Assume that there is a call connect charge of C monetary units, and a holding time charge of H monetary units per time unit. In addition, let U denote the loss of user utility (in monetary units) due to a setup delay when setting up a VC. Then, $C+U$ denotes the monetary cost of a call setup.

As an example, assume that a service provider charges 2 monetary units for call setup, and 1 monetary unit for every time unit that the call is kept open, irrespective of the traffic carried (there may be an additional charge per unit traffic, but that does not change our argument). Suppose that the call takes 1 time unit to setup due to propagation and call setup delays. Then, a manager may decide, as discussed above, that this additional delay has a monetary cost of 5 units. Thus, the true cost of opening a VC would be 7 monetary units. We will denote $C+U$ as O, the opening cost of a VC.

5.1 Holding policies

The critical parameter in deciding when to close a circuit is the ratio O to H, denoted Δ (measured in *time* units). Suppose that a packet has arrived at a VC at time t. If the next packet arrival is expected more than Δ in the future, then holding the VC till the next packet arrival is more expensive than closing it, and opening it again later. On the other hand, if the next packet is expected before Δ time units, we should hold the VC till the next arrival. Thus, any holding policy must estimate the next packet arrival, and close the VC if this exceeds Δ. We now describe three families of holding policies for this cost model.

The non-causal optimal scheme is trivial - if the next arrival is more than Δ time units in the future, the VC is closed, else it is kept open.

Any timer-based strategy for choosing a holding time must proceed as follows: keep the connection open for some time $T \geq 0$, and if no packet arrives, drop the connection. The only issue is the choice of T. We model the inter-arrival times as being sampled from some unknown underlying distribution χ, where each inter-arrival gap is an independent sample. The choice of T depends in a non-trivial way on χ. For instance, if we know that a packet will arrive within ϵ time with probability 1/2 and otherwise will arrive at time $t >> \Delta$ than the optimum strategy is to set $T = \epsilon$. The answer is completely different if χ is the uniform distribution with the same mean and standard deviation.

Since we are working only with the estimated mean and deviation, our knowledge of the distribution is limited. We denote the exponentially averaged value of the inter-arrival times as I (α was chosen to be 0.1). Let the estimated mean deviation from I be σ. We will try to model χ using I and σ.

It is likely that the bulk of the mass of χ lies in the interval $[I-2\sigma, I+2\sigma]$. So, if $I-2\sigma > \Delta$, then the VC should be closed immediately. Similarly, If $I+2\sigma < \Delta$, then the VC should be kept open, at least till $I+2\sigma$. If Δ lies in the interval $[I-2\sigma, I+2\sigma]$, then we have to make some assumptions about how the probability mass is distributed within the interval $[I-2\sigma, I+2\sigma]$. For simplicity, we assume that the mass is concentrated around the mean and so we close the circuit if $I > \Delta$ and keep it open otherwise. Thus, at the time of a packet arrival, we close the circuit if $I > \Delta$, and keep it open if $I < \Delta$. How long should the VC be kept open? The timeout value T is set to $max(I + 2\sigma, C)$ where C is is a cutoff parameter that ensures that even during a burst we keep the circuit open for a reasonable time. In our work, we chose $C = \Delta/5$. However, we have seen that the results are insensitive to choice of C in the range $\Delta/3$ to $\Delta/6$.

In our work, we estimate only the mean and standard deviation of χ. In recent work, Lund et al have shown that improved performance can be gained by maintaining a histogram of the distribution [4].

The third class of policies is LRU-based. We have already observed in Section 3.2 that LRU may be modeled as implicitly using a predictor which is a monotone function of the elapsed time since the last packet arrival. Thus, a simple predictor consistent with LRU is one which predicts the next arrival time of a packet to be c times the elapsed time since the last packet arrival, where $c > 0$ is any constant. Applying this method of predicting future arrivals we get a simple scheme: drop a conversation if it has been idle for Δ/c time units. This scheme, which we call the *LRU-delay based scheme*, has a nice property:

Lemma 1 *The cost incurred when using the LRU-delay based scheme with parameter c is no more than $max(c, 1/c) + 1$ times the optimal cost.*

To prove this, observe that the worst case input is one where the packets arrive Δ/c time units apart. The optimal cost of serving this sequence is $min(\Delta, \Delta/c)H$ per packet whereas our algorithm spends $(\Delta/c + \Delta)H$. Thus, we can guarantee that the cost incurred is no more than $max(c, 1/c) + 1$ times optimum.

Though the LRU-delay based scheme is straightforward, we would like to use more of the inter-arrival time history. To do so, we introduce a new concept, *the inter-reference interval to a level of an LRU stack.* We expect that due to temporal locality, the inter-reference interval to an LRU stack level will increase with the stack depth. Thus, maintaining statistics per stack level will show more stability than maintaining statistics per VC. Put another way, our intuition is that knowing that a VC has reached a particular stack level is more indicative of when the next packet is going to arrive at the VC than the per-VC statistics (which is what the timer-based approach uses).

Thus, in the *LRU-statistics scheme*, we maintain a per-level exponentially averaged mean inter-reference interval. When this mean is larger than Δ, any VC entering the level is automatically closed. We do maintain statistics for an infinitely large LRU stack, though, so that if a level that initially has a mean larger than Δ later attains a mean smaller than Δ, we automatically hold VCs that enter that level.

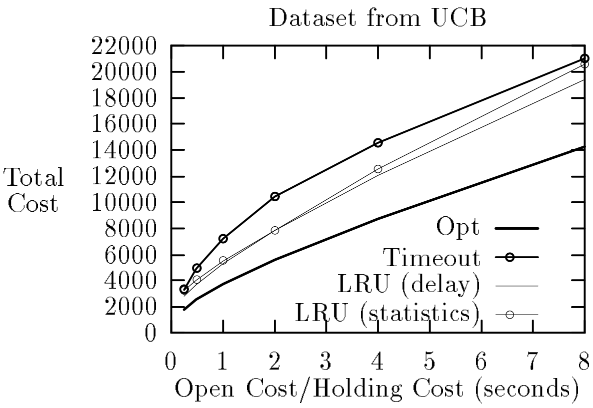

Figure 5: Total cost vs Δ: Holding cost model

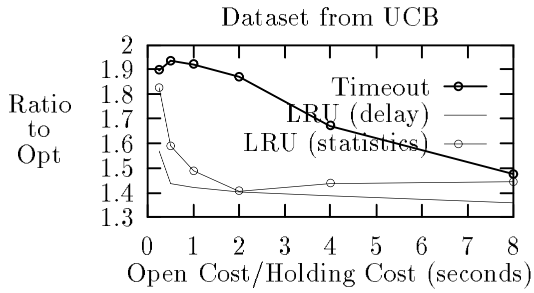

Figure 6: Normalized cost vs Δ: Holding cost model

5.2 Evaluation of policies

The various schemes mentioned above were simulated on each of the traces. The simulation technique was similar to that in Section 4.2 except that there is no limit to the number of open connections. Instead, we make the decision to keep a connection open independent of the other connections. For the LRU based schemes, we examine the reference stack after each packet is processed to check whether a connection needs to be dropped. For Timeout and non-causal optimum, after a packet is processed, we make a decision as to whether the corresponding connection is to be kept open or not. For Timeout we also maintain a list of active timeouts and service them.

Table 2: Relative performance: Holding cost model

Δ	Timeout	LRU (statistics)	LRU (delay)
0.25	1.76	1.78	1.68
0.50	1.82	1.71	1.54
1.00	1.87	1.59	1.48
2.00	1.75	1.53	1.47
4.00	1.63	1.52	1.46
8.00	1.52	1.39	1.42

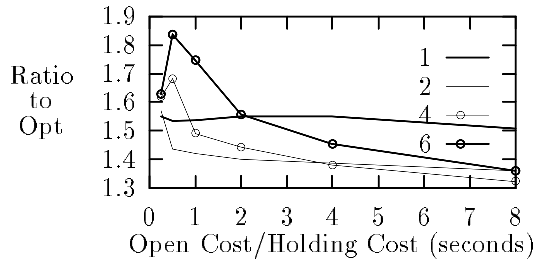

Figure 7: Sensitivity of LRU(delay) strategy to c: Holding cost model

We varied Δ from 0.25 to 8.0 and computed the total cost incurred for each dataset using each of the holding policies (here, for the LRU-delay based policy, we use $c = 2$). In Figure 5 we plot the variation in cost as Δ is varied for a sample dataset (the other datasets showed similar behavior). The cost increases as Δ is increased. This is to be expected as a larger Δ implies that each open costs much more. Even though the number of opens decrease when Δ is increased, the holding time costs go up while the costs associated with opens do not decrease. The non-causal strategy is significantly better than any of the others though the gap narrows as Δ decreases.

To compare relative performance of the strategies across different values of Δ and different datasets, we normalize the cost by dividing by the cost of the non-causal optimal strategy on the same data. A sample of the resulting values are shown in Figure 6. Table 2 lists the average taken over the all traces of the normalized cost of each policy. Timeout is consistently the worst, though its relative performance is not as bad as in the paging model. The LRU-delay based policy is the best, and LRU-statistics performs reasonably well. We see that the performance of LRU based policies improves as Δ is increased whereas that of Timeout first degrades and then improves.

The LRU-statistics scheme performs quite poorly for values of $\Delta < 1$ while in the range $1 \leq \Delta \leq 4$ the performance is within a few percent of the LRU-delay based scheme. For $\Delta \geq 8$ the LRU-statistics scheme is slightly better than the LRU-delay based scheme. The LRU-statistics scheme is always better than the timeout scheme, except for small Δ where the two schemes are comparable. A possible explanation for this behavior is that although the mean inter-reference gap at the current position of a VC on the LRU stack is a better predictor than the per-VC history, the prediction is effective only towards the tail of the LRU stack and is ineffective in the intermediate levels (as Δ is increased, the cutoff point on the LRU stack moves towards the lower levels.) Thus, for large Δ the LRU-statistics scheme may be preferred over LRU-delay. However, this must be balanced by the additional implementation complexity.

For the LRU-delay based policy, to arrive at an appropriate choice for c we studied the variation in performance for different values of c (Figure 7). We find that $c = 2$ (*i.e.* drop a conversation if it has been idle for $\Delta/2$ time units) works best. We notice that for $\Delta \geq 2$ the variation in performance is not large, the best strategy giving a ratio of 1.4 and the worst giving a ratio of 1.55 even though c is varied by a factor of 6. Although, this may seem surprising at first, there is a simple explanation. The LRU reference pattern indicates that initially there is a sharp drop in references as we move down the reference stack. After the first 10 levels the drop is much smoother. If we set the cut-off parameters in such a way that connections on the top 10 levels of the reference stack, which account for 90% of the references are undisturbed, and we drop connections beyond that, we will do quite well. Thus, we have considerable latitude in the intermediate portion of the reference stack since increasing the cut-off depth decreases open costs while increasing holding costs. These two costs tend to balance out, provided we do not go deep into the tail of the LRU reference curve.This feature has a nice consequence, that even if Δ is known imprecisely, the LRU-delay based strategy will work well.

6 Conclusions

We have done a detailed empirical examination of various holding time policies, assuming two different network pricing schemes. We propose a novel formulation of the holding time problem that incorporates the issue of network pricing. We collected data to empirically model the workload. The data shows temporal locality of reference in all traces. LRU based holding time policies use this temporal locality effectively and perform better than any other strategy examined. To our surprise, timer based schemes perform poorly. However, the LRU-delay scheme, which corresponds to a fixed system-wide timeout, is effective and trivial to implement. We have studied a number of variations of holding time policies and have found that none of the variations offer significant improvements in performance over the basic strategies discussed here.

7 Acknowledgements

We wish to thank Bruce Mah at UC Berkeley and Peter Danzig at the University of Southern California for helping collect the traces. Thanks also to Ramon Caceres at Matsushita Labs, who suggested the relevance of pricing in studying holding times, and made several useful comments on an earlier draft of this work.

References

[1] R. Caceres, "The Pyramid IP to X.25 Protocol Interface: Merging DDN and PDN Approaches", *Proc. Uniforum* (1987), Washington, DC, 1987.

[2] R. Caceres, P.B. Danzig, S. Jamin and D.J. Mitzel, "Characteristics of Wide-Area TCP/IP Conversations", *Proc. ACM SigComm*, September 1991.

[3] V. Jacobson, "Congestion Avoidance and Control", *Proc. ACM SigComm*, August 1988.

[4] C. Lund, S. Phillips, N. Reingold, "Adaptive Holding Policies for IP over ATM Networks", *Submitted to ACM SigComm 1994*, February 1994.