

AN ASSESSMENT OF STATE AND LOOKUP OVERHEAD IN ROUTERS

Deborah Estrin† Danny J. Mitzel††*

†Computer Science Department
University of Southern California
Los Angeles, CA 90089-0782
estrin@usc.edu mitzel@usc.edu

†Hughes Aircraft Company
Information Systems Division
P.O. Box 92919
Los Angeles, CA 90009

Abstract

The current Internet is based on a stateless (datagram) architecture. However, many recent proposals rely on the maintenance of state information within network routers, leading to our interest in the implications of a *stateful* network layer.

We collected internetwork traffic traces at the border routers of stub and transit networks, and use this data to evaluate, or predict, the effects of design alternatives for stateful architectures.

We present an estimate of the number of active conversations at a router, from this we derive the storage requirements for the associated conversation state table. Our analysis shows that at the network periphery fine grain control over the traffic may be possible. However, deeper within the network, it may be more efficient to manage the conversations at a coarser level.

We use our network traffic traces to perform trace driven simulations of an LRU cache, for different conversation granularities. Our results show that locality exists for each of the conversation types investigated. Using the Normalized Access Time function, introduced in [13], we show that improvements in state lookup time are possible with a small cache, even without special hardware.

1 Introduction

The current Internet is based on a stateless (datagram) architecture. However, many recent proposals rely on the maintenance of state information within network routers, leading to our interest in the implications of a *stateful* network layer. Proposed stateful mechanisms are designed to aid in congestion control [4, 23], provide greater control over routing decisions and network resources [20, 6], and support applications with specific bandwidth requirements

[21, 8, 1, 9, 10, 16, 14, 22]. The introduction of state information raises several concerns, among them are the storage requirements for this state information, and its efficient lookup during the packet forwarding process. These issues affect the ability of the proposed mechanisms to scale efficiently for use in very large, high speed networks.

In this paper we investigate systems that require the maintenance of state at the internetwork routers. We collected internetwork traffic traces at the border routers of stub and transit IP [18] networks, and use this data to evaluate, or predict, the effects of design alternatives, when implementing stateful architectures. We restrict our analysis to the virtual circuit oriented TCP [19] data traffic, since datagram traffic is inherently stateless.

An important design decision is the level at which conversations are defined. This determines the granularity of control over the network traffic, and affects the scalability of the system. Throughout this paper we look at several granularities of conversations, ranging from a single TCP application association, up to aggregation of all traffic between two communicating networks. We present an estimate of the number of active conversations at a router, from this we derive the storage requirements for the associated conversation state table. Our analysis shows that at the network periphery fine grain control over the traffic may be possible. However, deeper within the network, it may be more efficient to manage the conversations at a coarser level. Our analysis also shows the importance of developing an aggressive method for determining when a conversation has ended. This end-of-conversation detection must be accurate to maintain maximum efficiency. Deletion of state information too early may lead to expensive operations to recover the state, and also results in loss of historical information, which may be required for traffic control. However, the effects of storing idle state information, for even a few minutes, is shown to double the storage requirements in some cases.

In conventional IP, the only lookup function normally re-

* Authors names listed in alphabetical order.

10D.1.1

quired for packet forwarding is a routing table lookup. This has been recognized as a bottleneck in the forwarding process [7, 13]. It has been shown that the introduction of an LRU cache can substantially improve the efficiency of the packet forwarding process. Route caching is used in many existing routers. However, unlike the stateful schemes investigated here, which require lookup based on source-destination pairs, current route caches are based only on destination host or network. It is not intuitively obvious whether the solutions developed for routing table caches can be applied here. We use our network traffic traces to perform trace driven simulations of an LRU cache, for different conversation granularities. Our results show that locality does exist in each of the conversation types investigated. Using the Normalized Access Time function, introduced in [13], we also show that improvements in conversation state lookup times are possible using a small cache, even without special hardware.

The remainder of the paper is organized as follows. Section 2 details our data collection process, and the format of the collected data. An estimate of conversation table state requirements is included in Section 3, for several granularities of conversation definitions. Section 4 presents the results of simulations examining the performance of a LRU cache for conversation state table lookup. Section 5 summarizes the findings from our work.

2 Network Traffic Collection

Below we describe our data collection procedure, including collection sites, method, and trace data contents.

2.1 Traffic Collection Sites

We collected packet traces of actual wide-area network traffic for our investigation. At the University of Southern California (USC), USC Information Sciences Institute (ISI), and University of California Los Angeles (UCLA) we traced all traffic passing between the end sites and the Los Nettos regional network. This traffic includes all packets traveling between the campus networks and the NSFNET backbone, to other Internet sites. We characterize these as stub networks, they act only as a source or sink for all traffic. We believe these sites are typical of many stub networks in today's Internet, populated primarily by UNIX workstations and a handful of other systems.

Traffic captured at the CalTech Los Nettos-CERFNET (Los Nettos) router, and at the NSFNET's Ann Arbor Nodal Switching System (NSS), represents transit network traffic. The primary function of a transit network is to transport data between stub networks, and other transit networks. In our case, the networks can be further classified as regional and backbone transits. A regional network generally connects a collection of stub networks that are geographically close, and also acts as a gateway to the long-haul backbone. The NSFNET is a transcontinental backbone, with connections to regional networks throughout the United States.

2.2 Traffic Collection Methodology

At USC we used the NNStat program suite [2] to record all internetwork packets traversing the main campus backbone Ethernet. The collection routine was run on a dedicated Sun SparcServer 4/490, with the trace data being written directly to a local disk. During similar measurements, we estimated the packet loss rate by sampling a Poisson stream of ping packets. We observed that 3 to 5% of the ping packets were missing from the trace record.

For the data collection at ISI, UCLA, and Los Nettos, we used a collection program that we wrote. The collection program uses the Sun Network Interface Tap (NIT) in promiscuous mode, to monitor all packets on an attached Ethernet segment. Each of the trace collections were run on a dedicated Sun SparcStation 1+, with the trace data being written to a local 8mm Exabyte tape drive. The estimated packet loss rate for all trace collects was less than 1%.

Data collection at the NSFNET's Ann Arbor NSS used a modified version of the NNStat *statspy* program. The data collection was run on an IBM RT, at the NSS, which contains promiscuous token ring hardware. This IBM RT is dedicated to monitor traffic traversing the local node. Packets collected were written directly to a local disk. We were unable to estimate the packet loss rate in this environment.

All of the results presented throughout the paper are from post-processing the collected traces. This method was selected as it requires the least real-time processing during data collection, and provides the most flexibility for data analysis. However, it does require more storage than does on-the-fly aggregation.

2.3 Traffic Data Contents

Most stateful router mechanisms proposed, either include a specific connection identifier field in the packet header, or derive the identifier from existing header fields. IP does not contain an explicit connection identifier, however, the `<protocol, source address, source port, destination address, destination port>` tuple does define a unique association. We noted earlier that we only look at packets with the protocol field set to TCP, this leaves the remaining four fields for use in creating conversations definitions.

The trace records collected at USC, ISI, UCLA, and Los Nettos include a time stamp and the first 56 bytes of the network headers for each internetwork IP packet observed.¹ The time stamp records the arrival time of the packet at the network interface of the collection machine. The 56 bytes of header information includes the complete datalink (Ethernet), network (IP), and transport layer (UDP, TCP) header structures.

For the NSFNET collect, we recorded a time stamp and minimal subset of the network header fields required for our analysis, for each TCP packet observed. Again, the time

¹No information was collected from the user data portion of the packets, at any of the sites, and the host address fields were processed to protect the privacy of communication over the various networks.

stamp represents the time that the packet arrived at the collection machine's network interface. The subset of header fields saved, includes the IP source and destination addresses, and the TCP source and destination port numbers. Storage space limitations at the collection site required us to collect only TCP packets, to increase the duration of the trace.

Table 1 presents a summary of the network traffic collects, including starting time, duration, number of packets collected, and estimated packet loss rate. Throughout the remainder of the paper we present the detailed analysis only for the USC, Los Nettos, and NSFNET sites. We use the USC data to characterize stub network sites. From our analysis, we found that the other stubs exhibited similar characteristics, and the USC site provided the largest data set. We use the Los Nettos and NSFNET data to characterize transit networks, at the regional and backbone levels, respectively.

As noted in Table 1, the network traffic collects at the stub sites, and the Los Nettos regional, were each for 24 hours. We were careful to collect these traces during non-holiday weekdays, and believe they are representative of "normal" days. We are interested in network performance during "busy" periods, that is when efficient state management and lookup are most important. The 24 hour traces show that the number of active conversations is time dependent (see Figure 1). Heimlich [11] also noted similar time dependencies in network traffic on the NSFNET backbone. We were careful to collect our trace during the busy periods. The before-mentioned storage space limitations at the NSFNET site, limited the duration of our collect to about 35 minutes at that site. This shorter trace is still a useful data point for our analysis, since most conversations have a duration of much less than 35 minutes on the current Internet [5, 3].

3 Router State Requirements

This section discusses several of the alternatives for defining conversations based on the information available in the network (IP) and transport (TCP) level headers. We present six conversation types, and map these to proposed stateful mechanisms. We then examine how the conversation granularity affects the level of traffic control available, and estimate the router storage requirements for maintaining the conversation state information.

3.1 Stateful Router Mechanisms

Following we present an overview of several recently proposed mechanisms that we will examine throughout this section.

Fair Queueing (FQ) [4] emulates a bit-by-bit round robin service discipline. If N conversations are active, FQ attempts to allocate $1/N$ of the available bandwidth to each conversation. A *bid* is used to order the packet transmissions, it is calculated as the expected finishing round to service a packet, minus an offset (δ) to affect priority.

Flow Protocol [23] emulates a TDM system. Each conversation allocates bandwidth based on its expected Ar-

rival Rate (AR). Each packet arrival increments a per-conversation Virtual Clock (VC) by $1/AR$. The VC is then used to order the packet transmissions.

Visa Protocol [6] is designed to control access/use of resources in communicating Administrative Domains (AD). Before an inter-AD conversation between two hosts can be established, an exit and entrance *visa* must be obtained from an authorization server. Once a conversation is established, the visas are used to cryptographically sign each packet; access rights are checked at the borders of the two end ADs.

Inter-Domain Policy Routing (IDPR) [20] is a routing protocol incorporating robust policy enforcement controls. When setting up a new Policy Route (PR), policy constraints are checked at each AD border router along the entire route, from source to destination AD. To reduce overhead, it is possible to multiplex transport sessions over a single PR.

ST-II [21] is the Internet Stream Protocol, it is designed to coexist alongside IP to provide resource guarantees. It is based on streams, which are point-to-multipoint simplex paths. These are useful for supporting applications such as video teleconferencing. Most of the protocol functionality addresses stream setup and their management, simplifying the forwarding process for efficiency.

There are several other new proposals that rely on state in routers. These include Delay and Jitter Earliest-Due-Date [8, 22], designed to meet delay and jitter bounds for real-time channels; Asynchronous Time-Sharing [16], which is similar to ATM but explicitly includes a set of prioritized classes; and SRP [1], which is designed to support real-time communications using IP. Many proposals have also been developed for resource management in future ATM networks [9, 10, 14]. However ATM (similar to IDPR) may multiplex transport sessions over a single ATM circuit. Our subset of stateful mechanisms were selected because prototype implementations for most of them were available for study. Our results are directly applicable to such schemes by substituting their per-conversation overhead.

3.2 Conversation Models

Stateful router mechanisms differ in their concept of what constitutes a conversation. Demers et al. [4] discuss what should constitute a "user" for the Fair Queueing algorithm. They discuss how the granularity of the user definition can affect the "fairness" of the algorithm. In their case, it was decided that source-destination host pair conversations represented the best tradeoff between security and efficiency. In contrast, the IDPR routing protocol [20] controls access to network resources at the AD level. It defines conversations at the AD level, with finer grain control available, using optional policy constraints.

Site	Start Time	Duration	Internetwork		Loss Rate
			IP Pkts	TCP Pkts	
USC	Jan. 22, 1991 @ 14:24	24 hours	5,350,914	4,715,036	3 to 5%
ISI	April 4, 1991 @ 11:08	24 hours	2,346,624	1,260,336	< 1%
UCLA	May 12, 1991 @ 15:43	24 hours	2,277,072	1,697,348	< 1%
NSFNET	May 20, 1991 @ 09:00	35 minutes	502,111	502,111	unknown
Los Nettos	July 22, 1991 @ 09:51	24 hours	12,220,992	11,149,331	< 1%

Table 1: Traffic Collection Site Summary

HOSTdst	conversation defined based on destination host address
HOSTpair	conversation defined based on communication between a pair of hosts
HOSTpair+port	conversation defined based on the combination of communication between a pair of hosts, and the application port
NETdst	conversation defined based on destination network address
NETpair	conversation defined based on communication between a pair of networks
NETpair+port	conversation defined based on the combination of communication between a pair of networks, and the application port

Table 2: Conversation Types

The previous examples illustrate a range of granularities for defining conversations. We do not want to make our analysis specific to any single mechanism. For this reason, we define a set of conversation types, covering a range of granularities, which are then analyzed throughout the remainder of the paper (see Table 2). All of these conversation types can be directly derived from the packet headers collected, by examining the IP source and destination address fields, and the TCP source and destination port numbers. Throughout this section we discuss how these conversation types can be used to approximate the conversation definitions of many of the proposed stateful router mechanisms presented earlier.

The HOSTdst and NETdst conversation types aggregate all traffic traveling to a common host or network, into a single conversation. This is similar to what current IP route caches do. The HOSTpair conversation type performs aggregation between any two communicating hosts. Earlier it was pointed out that this definition is used by the Fair Queueing [4] and VISA [6] protocols.

Many of the most common wide-area TCP applications can be identified by their well known port [5, 3]. By looking into the TCP header, we can extract this port number, and determine the associated application. Adding the port to the HOSTpair conversation type, to obtain the HOSTpair+port conversation type, results in finer grain traffic control. Each conversation now aggregates simultaneous invocations of an application, between a pair of hosts. This aggregation is useful for approximating type of service (TOS) requirements, where multiple invocations of an application require similar performance guarantees. In today's Internet, the probability of a simultaneous invocation of an application between host pairs is quite small [5, 3]. Therefore, it may be possible, at

the current time, to further assume that each conversation at this level represents an individual application association.

The NETpair and NETpair+port conversation types are analogous to the host level conversations just discussed, except that they aggregate traffic at the network rather than host level. By making the simplifying assumption that network numbers approximate Administrative Domains in the current Internet, it is also possible to investigate mechanisms operating at the AD level, such as the IDPR routing protocol. Under this assumption, these conversation types represent communication between ADs, optionally with additional policy requirements based on application type.

3.3 Unidirectional vs. Bidirectional Conversations

Using the conversation types defined in the previous section, we also investigated the implications of unidirectional vs. bidirectional conversation definitions. As an example, using the HOSTpair conversation type, and the two hosts A and B , the unidirectional definition would treat the flows $A \rightarrow B$ and $B \rightarrow A$ as independent conversations, while the bidirectional definition aggregates the two. It should be noted, all of the conversation types defined can operate in either a unidirectional or bidirectional fashion, except the HOSTdst and NETdst types, which are inherently unidirectional.

In systems that perform explicit path setup, such as IDPR, ST-II [21], and Delay Earliest-Due-Date [8], it may be possible to guarantee that the same route is used in both directions. This simplifies the creation of bidirectional conversations. Other mechanisms such as Fair Queueing, which by themselves do not perform explicit path setup, and rely on

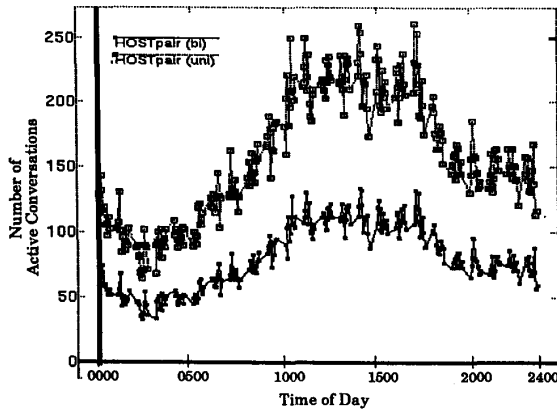


Figure 1: Number of Active Conversations for Bidirectional vs. Unidirectional Definitions (USC)

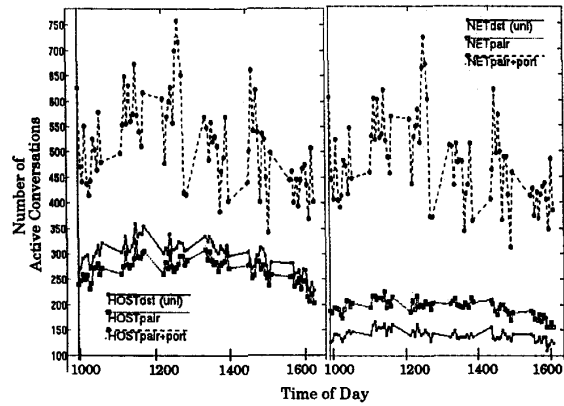


Figure 3: Number of Active Conversations (Los Nettos)

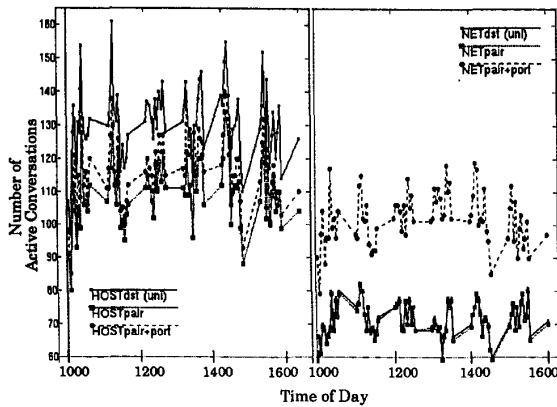


Figure 2: Number of Active Conversations (USC)

datagram routing, may be more appropriately modeled by unidirectional conversations.

Figure 1 shows the number of active conversations observed at USC over the course of the day, for both the unidirectional and bidirectional HOSTpair conversations types. All TCP conversations send packets in both directions, due to the ACK stream. However, most TCP applications actually transfer a substantial amount of data in both directions [5, 3], therefore, it is not surprising that the number of active conversations using the unidirectional definition is roughly twice that when using bidirectional conversations. Because of this simple relationship between the unidirectional and bidirectional definitions, during the rest of the state requirements analysis we present only the results for the bidirectional case.

3.4 Conversation Table Storage Requirements

Figures 2 thru 4 show the number of active conversations observed, over the course of the day, at USC, Los Nettos, and

NSFNET. A new conversation table entry is created whenever a conversation identifier is encountered that is not currently in the table.² The conversation is then reported as active until its state information is removed from the conversation table.³ The number of active conversations reported in the graphs, is the number of elements in the conversation state table, sampled at 5 minute reporting intervals.

All of the networks showed substantial increases in the number of conversations table entries that must be maintained, when going from coarse to finer grain conversation definitions. These ranged from doubling, in the case of USC, while a seven-fold increase was observed at times at the Los Nettos router. USC exhibits its largest change when going from the network level definitions, to the host level. This indicates that there are concurrent conversations at the network level, but between different hosts. This make sense, since most users are on individual workstations, rather than a few timesharing systems. The small change observed when adding the well known port to the conversation identifier, indicates that normally there is only a single application (i.e. Telnet, FTP, etc.) in use. The Los Nettos data shows a smaller percentage of change when going from the network level to host level conversation definitions, indicating simultaneous conversations are common at the network and host levels. However, there is a large change when adding the well known port to the conversation definition, indicating a large number of different applications in use between the communicating hosts. The NSFNET data shows that about 450 individual networks were communicating during the collect period, while the large change, when going from the NETdst to the NETpair conversation types, indicates that there are few simultaneous conversations between networks on the backbone.

No matter the reasons for the growth in number of active conversations for the finer grain definitions, this phenomena

²The conversion from TCP/IP packet header fields to conversation IDs was covered in Section 3.2.

³We use the heuristic of removing a conversation table entry after 5 minutes of inactivity.

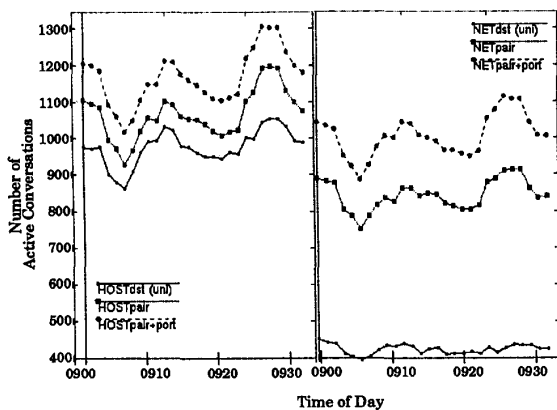


Figure 4: Number of Active Conversations (NSF)

has significant effect on the stateful mechanism. The finer grain definitions allow greater control over network traffic, but they also directly affect the storage requirements for the associated conversation state table. Most stateful router mechanisms are likely to store the conversation state table as some type of search tree or hash table. These table organizations incur a small amount of storage overhead for pointers or other table state information, however, the greatest factors determining table size are the number of entries and the size of each entry.

Table 3 estimates the conversation table storage requirements for several current prototype implementations of stateful router mechanisms. Each stateful mechanism is mapped to one of the defined conversation types, and then the storage requirements are calculated, based on the maximum number of entries observed in the conversation table, during the traffic traces. Most of the per entry data structure size estimates were obtained by examining actual implementations of the mechanisms, with the exception of Fair Queuing which was estimated using the description in [15].

As mentioned earlier, the IDPR routing protocol controls access to network resources at the Administrative Domain (AD) level. The mapping of IDPR to the NETpair conversation type was based on our simplifying assumption that a network address can be used to approximate an AD. IDPR also incorporates finer grain control, with the addition of type of service (TOS) constraints, user classes, and additional policy constraints. We have approximated the addition of TOS constraints by adding the well known port number to the NETpair conversation type. This assumes that each application type specifies common TOS constraints for all instances of the application. Both Fair Queuing and VISA operate at the host pair level, these map directly to the HOSTpair conversation type, while the Flow Protocol manages individual applications (mapping to HOSTpair+port). ST-II also operates at the application level, however it supports point-to-multipoint⁴

⁴ST-II also supports multicast communication, however, our data is for point-to-point connections, so this analysis does not consider mul-

simplex communication, therefore we have mapped it to the unidirectional conversation type (HOSTpair+port unidirectional).

Table 3 shows that all of the stateful mechanisms considered, with the exception of ST-II, have very modest storage requirements. Excluding ST-II, all of the stateful mechanisms require less than 10 Kbytes of state information at the stub network, and less than 80 Kbytes at the transit. ST-II requires the maintenance of much more state information per conversation (over six times that of any of the other mechanisms), and it also maps to the finest grain conversations type. It may not be possible to deploy ST-II into a very large internet for control of all conversations, however, ST-II is only required by those applications needing strict performance guarantees (such as packet video and voice). It may be possible to deploy it alongside IP, for those classes of applications requiring its functionality.

3.5 Aggregation of Conversations

Van Jacobson and Dave Clark recently proposed additional functions for IP routers, to provide resource management capabilities. Their scheme also introduces additional state information at the network routers. State is maintained based on *classes*, which are defined as aggregates of packets at different levels of granularity. These aggregate classes are defined based on policies at the router, and are organized into a hierarchy. This architecture allows policy decisions, such as enforcing resource constraints, to be enforced at a fine granularity (possibly on individual applications) before large amounts of network resources can be used. Deeper within the network, processing the different classes is simplified by operating upon higher level aggregates, thus reducing the total number of classes. Lazar's Asynchronous Time-Sharing [16] also defines classes, which aggregate conversations with similar performance requirements. His scheme differs in that classes are not defined hierarchically.

Our data shows that this type of architecture, incorporating a hierarchy of granularities, can improve the scalability of a stateful system, to perform efficiently in an environment such as today's Internet. In addition to the earlier arguments for this hierarchy, we see it as a means to dramatically reduce the number of conversation table entries maintained at the router. This reduction in the number of conversations table entries translates directly into smaller state requirements, and may also improve state lookup efficiency.

From Figure 2, we see that for the stub network, operating on the traffic at the application level (HOSTpair+port) resulted in a maximum of 150 active conversations entries. This does not seem excessive for the fine grain level of control provided. Coarser grain control, such as NETpair, can be applied to achieve a 45% reduction in the conversation table's size. However, because of the small number of active conversations, this translates into a reduction of only 68 table elements.

ticast support. Multicast could actually be applied to all of these protocols.

Stateful Mechanism	Conversation Type	Per Conv Entry Size	SITE	
			USC	NSFNET
IDPR (ADs only)	NETpair	56 bytes	4,592 bytes	51,128 bytes
IDPR (ADs+Policy)	NETpair+port	56 bytes	7,392 bytes	62,552 bytes
Fair Queueing	HOSTpair	16 bytes	2,144 bytes	19,184 bytes
VISA Protocol	HOSTpair	64 bytes	8,576 bytes	76,736 bytes
Flow Protocol	HOSTpair+port	38 bytes	5,700 bytes	49,628 bytes
ST-II	HOSTpair+port(uni)	392 bytes	114,464 bytes	852,600 bytes

Table 3: Conversation Table Storage Requirements

On the backbone network, exercising control over individual applications, results in greater than a ten-fold increase in the number of entries in the conversation state table, when compared to the stub network (see Figures 2 and 3). Changing to the coarser NETpair conversation type can result in a 30% reduction in the size of the conversation table. With the large number of active conversations at the backbone, this translates into a substantial reduction of nearly 400 table elements. However, this still leaves 900 elements in the table. Further reductions may be possible, and necessary, by using even coarser aggregations, such as combining multiple applications with similar performance constraints, or based on policy considerations.

3.6 Deleting Idle State Information

Throughout the conversation table state requirements analysis, we have used the heuristic of deleting a conversation from the state table after 5 minutes of inactivity. In Figure 5, we present the distribution of conversation durations using two different end-of-conversation heuristics, for the USC data using the HOSTpair conversation type. The graph shows that, in fact, the average conversation duration is much shorter; on the order of 15 seconds.⁵ We chose the 5 minute timeout to encompass approximately 90% of the conversations.⁶

Determining when a conversation has ended, and freeing up its associated state information is an important and also non-trivial task. In the simplest case, where each conversation might be equated to a single TCP connection, a separate state machine could be maintained for each conversation transiting the router. The router could then accurately determine when a connection had been torn down, and release its local conversation table entry. However, this would most likely be too expensive both in terms of computing and storage resources at the router. The process becomes even more difficult when the conversations aggregate multiple TCP connections.

As mentioned, using 5 minutes of silence to mark the end of a conversation is just a heuristic. Figure 5 also shows the observed conversation durations using a second end-of-conversation heuristic, one which uses 15 minutes of silence.

⁵Conversation duration is defined as the time during which the conversation was active, not the length of time the entry was in the conversation table (which includes the timeout period). It is calculated as $Timestamp_{lastpkt} - Timestamp_{firstpkt}$.

⁶Caceras et al. performed conversation analysis based on 5 minutes and 20 minutes of silence, they found no significant differences using the shorter interval.

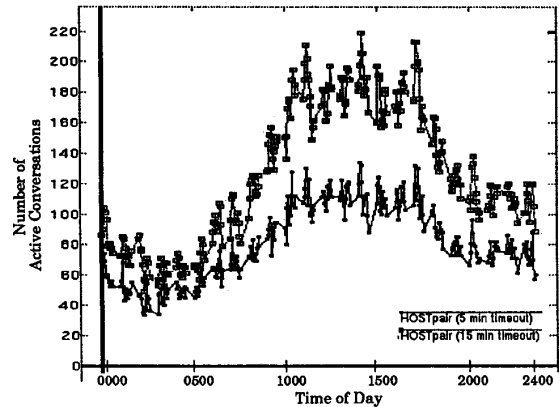


Figure 5: Number of Active Conversations for 5 Min. vs. 15 Min. Marks End-of-Conv. (USC)

The graph shows a small discrepancy between the two heuristics, indicating that the 5 minute heuristic may error, by removing the conversation's state information when in fact it is still active, just in an extended idle period. This may be expensive if the state recovery mechanism must then be invoked, or if traffic control information is lost.

The preceding argument about the problems with the 5 minute heuristic may seem to argue for the 15 minutes of silence heuristic, however, Figure 6 shows the dangers in maintaining idle table entries for too long. Figure 6 shows the number of entries that must be maintained in the conversation table, for both the 5 minutes and 15 minute heuristics, for the USC data using the HOSTpair conversation type. The graph shows a 65% increase in the peak number of conversation table entries when the less aggressive 15 minute silence interval is used (i.e., from 130 to 215). This large increase in number of conversation table entries translates directly into larger state tables at the router, and places additional limitations on scalability of the stateful mechanisms.

Additional investigation in this area is probably appropriate to develop a better understanding of the problem, and the effect of the 5 minute heuristic on our simulation results. It may be possible to develop better heuristics using knowledge of the individual applications or conversation types. We anticipate conversation durations to increase in the future, when longer duration voice and video connections become more popular. This may also affect our results, and

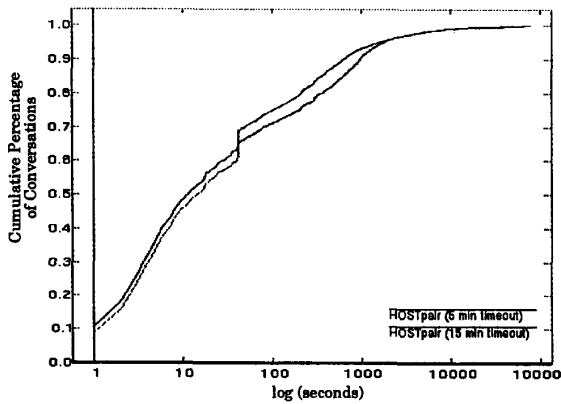


Figure 6: Conversation Durations for 5 Min. vs. 15 Min. Marks End-of-Conv. (USC)

argues for the development of a more dynamic or adaptable method for detecting the end of conversations. It should also be noted that this is not a problem in systems such as ST-II or Flow Protocol, where explicit setup and teardown is performed. IDPR also performs explicit setup and teardown, however it may keep routes installed for longer periods in the hope that they'll be reused. It may require some cooperation between the endpoints and the router to manage the release of conversation table entries to reduce the amount of idle state information maintained.

4 Conversation State Lookup

Conversation state lookup may be a bottleneck for these stateful mechanisms, especially when operating within the time constraints introduced by very high speed networks. In this section we look at whether caching schemes might be appropriate to increase the efficiency of the lookup process. First we determine whether locality is exhibited for each of the conversation types. Next we look at cache performance. Last, we compare the performance of the systems with and without a cache.

4.1 Locality Analysis

The success of a caching scheme is predicated on the fact that locality exists in the reference string. When looking at packets traversing a router, it makes the most sense to look at the temporal locality of successive packets. Temporal locality implies there is a high probability of reuse of an address.

There are several ways to quantify a measure of locality. Heimlich [11] measured the probability that a packet referenced the same address as the previous packet. Jain [13] used the LRU Stack Model to measure the probability of reference to different stack locations. The LRU stack model arranges the addresses from most to least recently seen. Whenever an address is seen, it is moved to the top of the stack. In a reference string with high temporal locality, we expect to see a

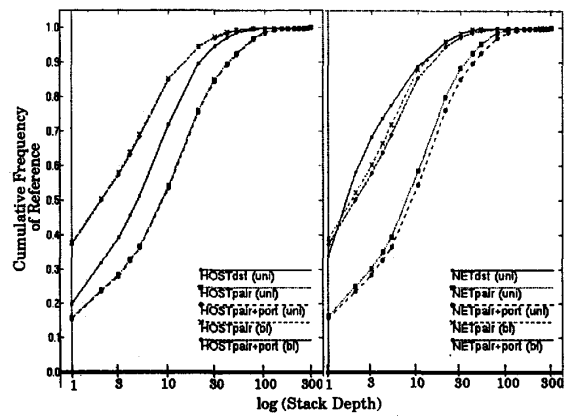


Figure 7: Stack Reference Depth Cumulative Probability (USC)

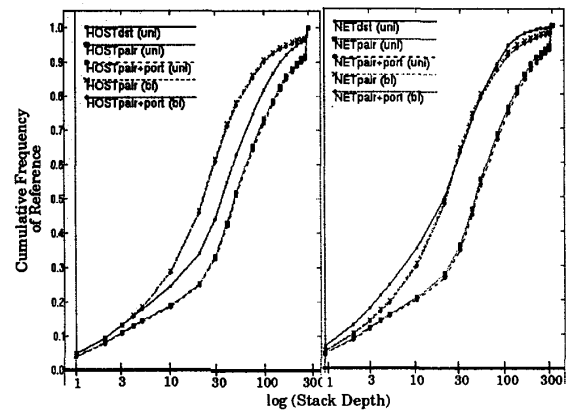


Figure 8: Stack Reference Depth Cumulative Probability (NSF)

high probability of reference to the top stack position. Figures 7 and 8 show the cumulative probabilities of referencing different stack locations for the USC and NSFNET networks.

We can quantify the temporal locality by looking at the probability of referencing the top stack position (p_1). We see that for the stub network the temporal locality is fairly high (38%), while the transit backbone shows much less locality (4%) due to the larger number of active conversations. Even with the low probability of back-to-back packets belonging to the same conversation in the transit network, it appears that a caching strategy could be effective, if the cache is of sufficient size. For all but the unidirectional transit conversations, over 90% of the references are found within the top 100 stack positions.

It is interesting to note that the bidirectional host and network level conversations show at least as much locality as the host and network destination only conversations. The destination only conversation types are used in current route caches, so we may expect the bidirectional conversations to perform even better than current route caches.

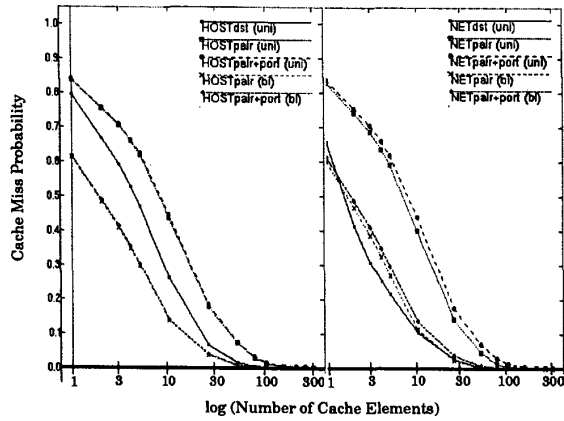


Figure 9: LRU Cache Performance (USC)

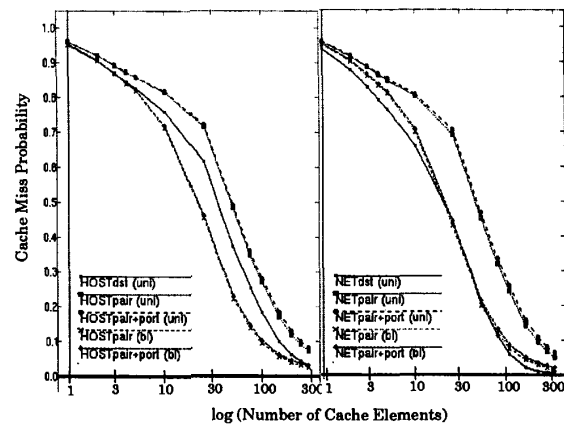


Figure 10: LRU Cache Performance (NSF)

Also note the large difference between the bidirectional and unidirectional conversation types, this reflects the bidirectional nature of TCP connections. At the stub, where there are fewer active conversations and it is more likely that back-to-back packets for the same conversation will be seen. We see that when going from the bidirectional to unidirectional conversation definitions, the locality is initially reduced by over 50%. This results from counting packets traveling in opposite directions between the same source-destination pair (i.e., part of the same TCP association) as separate conversations. The initial effects are less pronounced in the transit network due to the much lower probability of back-to-back packets for a conversation. However, using a stack size of greater than 5, we see the probability of encountering another packet for a bidirectional conversation has grown to 20%, and the bidirectional TCP traffic once again begins to interfere in the unidirectional conversations. With larger stack sizes, this interference becomes as pronounced as in the stub network.

4.2 LRU Cache Performance

The degree of temporal locality shown in the previous section implies that a caching strategy may be effective in increasing the efficiency of the state lookup process. In this section we look at cache performance. We used the traffic traces previously collected to drive our simulations. An LRU cache was assumed, based on the findings of Jain [13] and Feldmeier [7]. Cache size was varied from one to 300 elements. Since we are interested in steady state performance, we processed the first 50,000 packets from each trace to initialize the cache. After that point, we maintained a count of the number of packets forwarded, and the number of cache misses.

Figures 9 and 10 show the cache performance for a conversation type is directly related to the degree of locality exhibited (as shown in Figures 7 and 8). We see, as anticipated, that the bidirectional conversation types perform slightly better than current route caches, except for the stub network using the network level cache. We also see the pattern of only

a small difference among the four bidirectional conversation types, and similarly among the four unidirectional conversation types; while there is a substantial difference between the two sets. The transit network shows much larger miss rates than the stub for a small cache. However, 80 to 90% cache hit rates are still possible with a moderate sized cache (50 to 100 elements).

4.3 Normalized Access Time

Depending upon the cache organization it is possible to develop the notion of an optimal cache size. As the cache size increases, the probability of a cache miss decreases. This seems to argue that an optimal cache be as large as possible. However, as the cache size increases, the search time may also increase. Jain [13] suggests the Normalized Access Time to determine the optimal balance between miss rate and search time.

$$\text{Normalized Access Time} = \frac{\text{Search Time w/Cache}}{\text{Search Time w/o Cache}} = \frac{T_c + pT_{st}}{T_{st}}$$

where

T_c = Cache Lookup Time

p = Cache Miss Probability

T_{st} = State Table Lookup Time

Using this formula, the optimal cache size is obtained at the minima of the function, over the range of cache sizes. In the remainder of this section, we assume that the state table can be searched in $O(\log_2(n))$ time, where n is the number of entries in the table. This makes $T_{st} = 1 + \log_2(n)$. All that remains is to make an assumption about the cache access time.

We first look at the case when the cache access time and organization are identical to that of the state table, giving $T_c = 1 + \log_2(c)$. This implies that performance improvements are obtained solely from exploiting the locality in the references. Figures 11 and 12 show the results for stub and transit networks. The stub network shows an improvement of

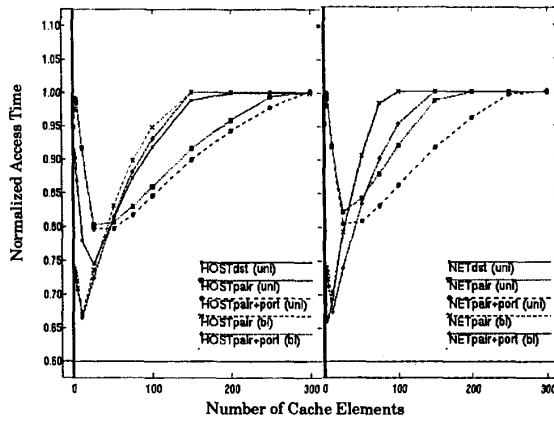


Figure 11: Normalized Access Time for LRU Cache (USC)

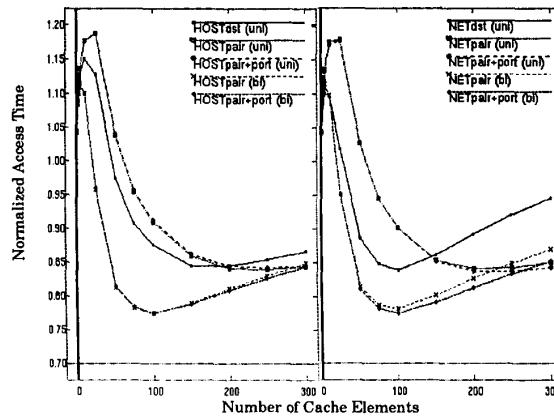


Figure 12: Normalized Access Time for LRU Cache (NSF)

30 to 35% at the optimal size of 10 elements for bidirectional conversations, compared to a 25% improvement for current route caches. The maximum improvement for the other unidirectional conversations is only 20% with a cache size of 25 elements. The transit backbone network shows slightly lower gains for both sets of conversation definitions, and the optimal cache sizes are much larger. Again we see about a 10% improvement in the performance of the bidirectional conversations, over current route caches. We also see that for small cache sizes (< 10), the state lookup performance with a cache may actually be worse than without a cache.

The previous results show that performance improvements are possible with a cache that exploits only reference locality. There are other cache designs that represent tradeoffs between ease of implementation and cost. A very simple implementation is to store the cache as a doubly linked list in memory. However, this requires a linear search, giving $T_c = c/2$. Another alternative is to incorporate a hardware cache. We assume a fully associative static RAM implementation with 25ns access time, and a 100ns main memory access time, giv-

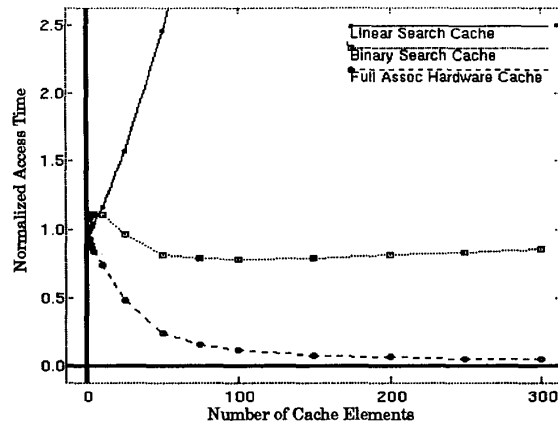


Figure 13: Normalized Access Time for Cache Architecture Alternatives (NSF)

ing $T_c = 0.25$. Figure 13 shows the normalized access times for the transit network for these cache design alternatives.

It's apparent that the linked list implementation can be ruled out immediately. For caches of size 1 or 2 it shows a slight improvement (1%), and its performance quickly degrades. Both the binary search table, and hardware cache are viable solutions. The hardware cache has constant access time; as the cache size increases and the miss probability decreases, we see the normalized access time $\gg 0$. Again, this seems to argue for as large a cache as possible, but current technology and costs will effectively set an upper bound on this solution. The in-memory binary search table solution is the same as shown earlier. It represents a compromise, providing moderate performance gains, with no requirement for additional hardware and its associated costs.

5 Conclusions

Future internet protocols will support a range of data services, some of which will require a departure from the original, stateless, IP router architecture. This paper is a first attempt to empirically quantify the state and caching requirements implied by stateful protocols.

We showed, using traffic traces from stub, regional, and backbone networks, that the maximum conversation table size is surprising low, given the size of the current Internet. Much of this has to do with the very short conversation durations observed. However, we believe that as new applications, such as voice and video, become more popular, the average conversation durations observed will increase. This will have the effect of increasing the likelihood of simultaneous conversations, and thus increase the number of elements, and state requirements for the conversation tables.

As we approach gigabit speeds on the backbones, it will be necessary to reduce the bottlenecks in the packet forwarding process. We have shown that it may be possible to improve performance using a simple LRU cache, similar to that used

in current route caches. Other caching schemes and optimizations are the subject of further study.

6 Acknowledgments

We wish to acknowledge the significant contributions made by related traffic analysis efforts of Peter Danzig and Sugih Jamin. In addition, we are very grateful to the people who authorized and assisted our data collections: Steve Wolf at NSF; Hans-Werner Braun, and Bilal Chinooy at Merit; Mark Brown and John Conti at USC; Bob Braden, Walt Prue, and Jim Koda at ISI; Rich Wales at UCLA; and Heather Sherman at CalTech. Lixia Zhang and Lee Breslau read earlier drafts of the paper, and along with the referees, provided many valuable comments.

References

- [1] Anderson, D.P., Herrtwich, R.G., Schaefer, C., "SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet," UCB TR-90-006, February 1990.
- [2] Braden, B., DeSchon, A.L., *NNStat: Internet Statistics Collection Package: Introduction and User Guide*, ISI Research Report ISI/RR-88-206, August 1988.
- [3] Danzig, P.B., et. al., "An Empirical Model for Simulating Wide-Area TCP/IP Conversations," to appear in *Journal of Internetworking*.
- [4] Demers, A., Keshav, S., Shenker, S., "Analysis and Simulation of a Fair Queueing Algorithm," *Proceedings ACM SIGCOMM '89*, pp. 2-12.
- [5] Caceras, R., Danzig, P.B., Jamin, S., Mitzel, D.J., "Characteristics of Wide-Area TCP/IP Conversations," *Proceedings ACM SIGCOMM '91*, September 1991.
- [6] Estrin, D., Mogul, J., Tsudik, G., Anand, K., *Visa Protocols for Controlling Inter-Organizational Datagram Flow*, IEEE JSAC, May 1989.
- [7] Feldmeier, D., "Improving Gateway Performance with a Routing-Table Cache," *Proceedings IEEE INFOCOM '88*, March 1988.
- [8] Ferrari, D., Verma, D.C., "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE JSAC*, 8:3, April 1990.
- [9] Golestani, S.J., "Duration-Limited Statistical Multiplexing of Delay-Sensitive Traffic in Packet Networks," *Proceedings IEEE INFOCOMM '91*, pp. 323-332.
- [10] Guerin, R., Ahmadi, H., Naghshineh, M., "Equivalent Capacity and Its Application to Bandwidth Allocation in High-Speed Networks," *7th ITC Seminar*, Morristown, NJ, October 1990.
- [11] Heimlich, H., "Traffic Characterization of the NSFNET National Backbone," *Proceedings Winter USENIX '89*.
- [12] Jain, R., Routhier, S., "Packet Trains—Measurements and a New Model for Computer Network Traffic," *IEEE JSAC*, September 1986.
- [13] Jain, R., *Characteristics of Destination Address Locality in Computer Networks: A Comparison of Caching Schemes*, DEC-TR-592, February 1989.
- [14] Kalmanek, C.R., Kanakia, H., Keshav, S., "Rate Controlled Servers for Very High-Speed Networks," *IEEE Global Telecommunications Conference*, San Diego, CA, December 1990.
- [15] Keshav, S., "On the Efficient Implementation of Fair Queueing," *Proceedings ACM SIGCOMM '90*.
- [16] Lazar, A.A., Pacifici, G., "Control of Resources in Broadband Networks with Quality of Service Guarantees," *IEEE Communications Magazine*, September 1991.
- [17] McKenney, P.E., "Stochastic Fairness Queueing," *Proceedings IEEE INFOCOM '90*, August 1990.
- [18] Postel, J.B., *Internet Protocol*, RFC-791, September 1981.
- [19] Postel, J.B., *Transmission Control Protocol*, RFC-793, September 1981.
- [20] Steenstrup, M., *Inter-Domain Policy Routing Protocol Specification and Usage: Version 1*, BBN Technical Report No. 7346, July 1990.
- [21] Topolcic, C., *Experimental Internet Stream Protocol, version 2 (ST II)*, RFC-1190, October 1990.
- [22] Verma, D., Zhang, H., Ferrari, D., "Guaranteeing Delay Jitter Bounds in Packet Switching Networks," *Proceedings Tricomm '91*, Chapel Hill, North Carolina, April 1991.
- [23] Zhang, L., "Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks," *Proceedings ACM SIGCOMM '90*, pp. 19-29.