Efficient Demultiplexing of Incoming TCP Packets
SQN TR92-01

Paul E. McKenney (pmckenne@us.ibm.com)
Ken F. Dove

Sequent Computer Systems, Inc.
15450 SW Koll Parkway
Beaverton, OR 97006

Abstract

When a transport protocol segment arrives at a receiving system, the receiving system must determine which application is to receive the protocol segment. This decision is typically made by looking up a protocol control block (PCB) for the segment, based on information in the segment's header. PCB lookup (a form of demultiplexing) is typically one of the more expensive operations in handling inbound protocol segments [Fel90].

Many recent protocol optimizations for the Transmission Control Protocol (TCP) [Jac88] assume that a large component of TCP traffic is bulk-data transfers, which result in packet trains [JR86]. If packet trains are prevalent, there is a high likelihood that the next TCP segment is en route to the same application (i.e. uses the same PCB) as the previous TCP segment. In these environments a very simple one-PCB cache like those used in BSD systems yields very high cache hit rates.

However, there are classes of applications that do not form packet trains, and these applications do not perform well with a one-PCB cache. Examples of such applications are quite common in the area of heads-down data entry into on-line transaction-processing (OLTP) systems. OLTP systems make heavy use of computer communications networks and have large aggregate-packet-rates but are also characterized by large numbers of connections, low per-connection packet rates, and rather small packets. This combination of characteristcs results in a very low incidence of packet trains.

This paper uses a simple analytic approach to examine how different PCB lookup schemes perform with OLTP traffic. One scheme is shown to work an order of magnitude better for OLTP traffic than the one-PCB cache approach while still maintaining good performance for packet-train traffic.

1 Introduction

A Transmission Control Protocol (TCP) [Pos81] protocol control block (PCB) contains state information for one endpoint of a given connection. A TCP demultiplexing (*a.k.a.* PCB-lookup) algorithm must find the PCB corresponding to the connection for each newly-arrived TCP packet. The algorithm does this by mapping the packet's source and destination Internet Protocol (IP) addresses and TCP ports to the proper PCB. Since the addresses and ports total 96 bits, simple indexing schemes are not feasible; more complex schemes must instead be used.

A simple PCB management approach uses a simple, linear linked list of PCBs. This approach was used in the initial BSD system. Sequent's initial TCP implementation, based on BSD's, also used a simple linked list. During 1988, Sequent began designing a parallel implementation of TCP for its second-generation PTX operating system [Gar90]. An explicit goal of this effort was to support thousands of concurrent users connected by local-area networks. This effort resulted in, among other things, the algorithm described in Section 3.4.

About the same time, Van Jacobson was conducting research aimed at increasing TCP's single stream performance. As a result of this research, the BSD 4.3-Reno release augmented the linked list with a single-line cache referencing the last PCB found. This simple optimization has proven very effective in many environments; it was quickly incorporated into Sequent's algorithm because of its greatly-improved handling of bulk-data transfers.

Recently, the Transaction Processing Council published the TPC/A online-transaction-processing benchmark [Gra91], which is now in common use by database software and platform vendors. This benchmark is a very important development, because it provides a precise and realistic definition of an important application that requires large numbers of connections. In particular, the TPC/A benchmark allows objective comparisons of the effects of different protocols and algorithms.

At first glance, the definition of TPC/A is such that simulation must be used to analyze it. However, this article shows that there are simple, analytic expressions that give good approximations to the cost of PCB lookup imposed by the TPC/A benchmark.

Analysis based on TPC/A shows that this benchmark causes the BSD algorithm to perform very poorly. Similar analysis predicts that an algorithm suggested by Jon Crowcroft [Cro91][1] and another proposed by Craig Partridge and Stephen Pink [PP91] should achieve significantly higher performance under TPC/A, and finally that an algorithm used by Sequent [Dov90] achieves an additional order of magnitude improvement under TPC/A while maintaining good performance in other situations. This increase in performance has protocol-design implications, since it greatly reduces the need to add protocol mechanisms (such as connection IDs) that eliminate the need to search for PCBs.

Section 2 gives an overview of the communications behavior of the TPC/A benchmark; Section 3 analyzes the behavior of the above algorithms, and Section 4 presents conclusions.

2 TPC/A Benchmark

The TPC/A benchmark simulates a banking system in which customers make randomly-generated deposits to and withdrawals from a bank with several branches. The benchmark contains scaling rules that protect against "trivial" benchmark results being issued; these rules require (for example) that the size of the various elements of the database increase with increasing transaction rate.

The most important rule from a communications standpoint is that the number of users represented in the benchmark be at least ten times the transaction rate. Specifically, a 200 TPC/A TPS benchmark run must have at least 2,000 simulated users. The TPC/A rules are quite strict about how users must be simulated; in particular, the network load must faithfully represent that of real users.

Each simulated user does the following repeatedly:

1. Enters a transaction.
2. Waits for the response. The time between steps 1 and 2 is called the "response time." The response time for at least 90% of the transactions must be no greater than two seconds in order for the benchmark to be valid.
3. Waits for a randomly-selected period of time before returning to step 1. This time is called "think time." The think time is selected from a truncated negative-exponential distribution whose mean must be at least 10 seconds and whose maximum value must be at least 10 times the mean value. The purpose of think time is to simulate real-life delay from human data-entry personnel.

Therefore, the average time required for a user to enter a transaction will be at least 10 seconds, which is consistent with the rule that a given transaction rate must have at least ten times that many users.

3 Analysis

The preceeding description of TPC/A allows us to calculate the hit rates and miss penalties for PCB-lookup algorithms. In each of the following sections, we assume optimal use of the communications media. Each transaction requires four packets: (1) the query, (2) the transport-level acknowledgement for the query, (3) the response, and (4) the transport-level acknowledgement for the response.[2]

We will model the think time as if it were a true (rather than truncated) negative-exponential distribution. Since the truncation occurs only for values at least ten times the mean, this will have negligible effects on the results. In particular, only 0.004% of the values are neglected on average, and they sum to less than 0.4% of the total think time.

We also assume that a user can issue transactions that are spaced arbitrarily closely. In reality, a user may not issue a new transaction until he has received the response from his previous transaction. Typical TPC/A runs have fewer than 10% of the users waiting for a response at any one time, and as we shall see, the differences between the algorithms far exceeds this amount. This assumption is crucial; omitting it would require the analysis to carry enough state to determine, at any time, how many users are waiting for responses and consequently are unable to enter new transactions. The resulting state space for a 2,000-user system would be truly enormous (even given the low memory costs that we now enjoy).

[1]This algorithm was independently put forward by Gary Delp.

[2]Although delayed acknowledgments can eliminate the need for the second packet, this will have no effect on the results at the database server since this packet will be received only by a client.

The figure of merit is the expected number of PCBs searched. This is especially appropriate for large numbers of connections, because all the PCBs will not fit into contemporary on-chip caches. Since memory speeds and bandwidths have been and are expected to continue increasing much more slowly than CPU speeds [HJ91, SC91], moving the PCBs between main memory and the on-chip cache is and will continue to be the primary bottleneck. Hence, the number of PCBs examined is a very good surrogate for the time required to find the right PCB.

Since the negative exponential distribution is memoryless, each of the 2,000 users are equally likely to enter the next transaction. The memoryless property is discussed at length in any text on stochastic modelling (for example, "Introduction to Operations Research" by Hillier and Lieberman [HL86]). Memoryless means that the result of any trial is independent of past history. An example of a physical process that results in a distribution with the memoryless property is rolling a fair die and counting the number of rolls until a six appears.[3]

The following sections analyze the average cost of the BSD algorithm, the "move to front" algorithm proposed by Jon Crowcroft, the last-sender/last-receiver algorithm proposed by Craig Partridge and Stephen Pink, the algorithm in use in Sequent's TCP/IP product, and combinations of these algorithms.

3.1 BSD

BSD searches a simple linear list of PCBs, with a single-entry cache containing the PCB last found. Figure 1 shows a schematic of this list just after the arrival of a packet for the connections corresponding to PCB "B."

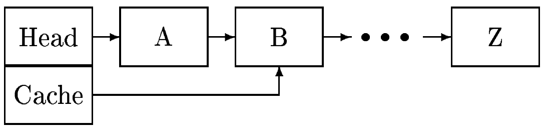

Figure 1: BSD PCB List

The hit rate for the PCB cache is $1/N$, which is 0.05% for a 200 TPC/A TPS benchmark. The average cost of a miss is a linear search scanning up to 2,000 PCBs. The average number of PCBs that must be examined is just one if we hit the cache and an additional $(N+1)/2$ if we miss. The probability of a hit is just $1/N$, so the probability of a miss is $(N-1)/N$. Thus:

$$C_{\mathrm{BSD}}(N) = 1 + \frac{N^2-1}{2N} , \qquad (1)$$

approaching $N/2$ for large N. This equation yields an average cost of a linear scan of 1,001 PCBs for a 200 TPC/A TPS benchmark. Since this is exactly the cost of a miss to three places, the cache is clearly providing little help. Because the data contained in all 2,000 PCBs will not fit into on-chip data caches for any currently-available microprocessor of which we are aware, this scan will involve traffic at least to an off-chip cache. In many systems, the scan will require accesses to real memory. This overhead motivates use of a different algorithm.

One might expect that there would be some small chance that the packets representing a transaction entry and the transport-level acknowledgement for the response might form a packet train, so that the proper PCB would be cached when the acknowledgement arrived. One would be right. There is a *very* small chance of this; the probability is about 1.9×10^{-35} for a relatively fast 200-millisecond response time in a 200 TPC/A TPS benchmark.[4] Keep in mind that the response time includes full database lookup, processing, commit, and logging for the transaction as well as the relatively small communications overhead, which itself includes a round trip time to the client machine. Thus, the average cost for the transaction-level acknowledgement will be about 1001 PCBs.

Although the BSD algorithm has served admirably in many common situations [Mog91], it appears safe to say that it was not designed with high-end online transaction processing needs in mind.

3.2 "Move to front" list

Jon Crowcroft proposed maintaining a linear list with a "move to front" heuristic; namely, when a PCB is found, it is moved to the front of the linear list. Figure 2 gives a schematic of the list just before arrival of a packet on connection "B" and Figure 3 gives a schematic of the list just after the arrival. Note that PCB "B" has been pulled to the front of the list.

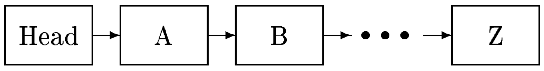

Figure 2: Crowcroft's PCB List Before Arrival

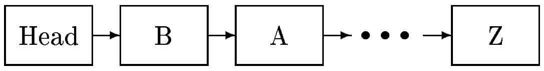

Figure 3: Crowcroft's PCB List After Arrival

[3]This process would result in a geometric distribution. The time required to see a single particular face of a fair die with an infinite number of sides that was rolled infinitely quickly would be exponentially distributed.

[4]Although there is a 96% probability that any given user will not offer a transaction or deliver a transport-level acknowledgement to a response during a given 200-millisecond interval, the probability that none of the 1,999 other users will not do so is indeed remote.

This heuristic has two consequences: 1) a slight increase in the number of PCBs searched for the TCP packet representing the entry of a new transaction and 2) a substantial decrease in the number of PCBs searched for the transport-level acknowledgement to the TCP packet representing the response. The decrease results in a significant overall reduction in overhead compared to the BSD algorithm.

Typically, many of the other users will have entered a transaction during a given user's (call him Jon) think-time interval. Thus, these other users' PCBs will precede Jon's in the list. Analytically, the probability of any given user having entered at least one transaction during an interval of time T is

$$F(T) = 1 - e^{-aT} \ , \tag{2}$$

where a is the per-user average transaction rate of 0.1 transactions per second. This expression is just the cumulative distribution function for the exponential distribution. The expected number of users from a total of (N - 1) users (all of them but Jon) to enter at least one transaction during this time will be

$$N(T) = \sum_{i=0}^{N-1} i \binom{N-1}{i} \left(1 - e^{-aT}\right)^i e^{-aT(N-1-i)} \ . \tag{3}$$

The i factor gives the number of users preceding Jon, the binomial factor gives number of different groups of i users that can be formed out of the $N - 1$ users other than Jon, $\left(1 - e^{-aT}\right)^i$ is the probability that those i users will precede Jon, and $e^{-aT(N-1-i)}$ is the probability that the rest of the users will follow Jon. Multiplying all of these together and summing over i results in a weighted average (the "i"s are being averaged, the rest of the factors comprise the weight) that gives the expected number of users that will precede Jon. Figure 4 shows a plot of Equation 3 for 2,000 users.

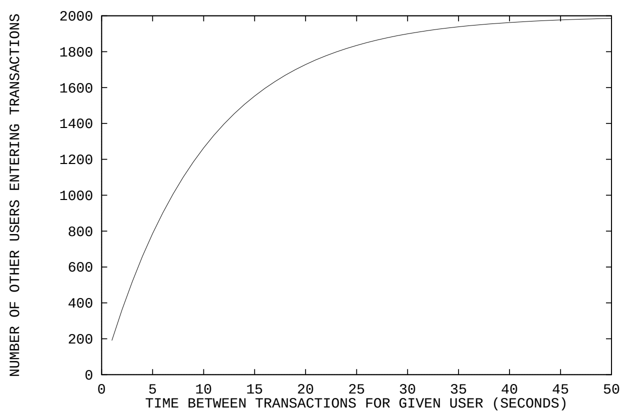

Figure 4: $N(T)$ for 2,000 TPC/A Users

Now, the probability that Jon's think time will be within an interval of width dT which is centered around a time T is approximately

$$f(T) = ae^{-aT} dT \ . \tag{4}$$

The approximation gets better as dT gets smaller. This expression is just the distribution function for the exponential distribution (if you ignore the dT for now).

If the think time T between the transport-level acknowledgement to the response to Jon's last transaction and Jon's current transaction is greater than the response time R, we have the situation shown in Figure 5. Any packet that arrives during the interval T

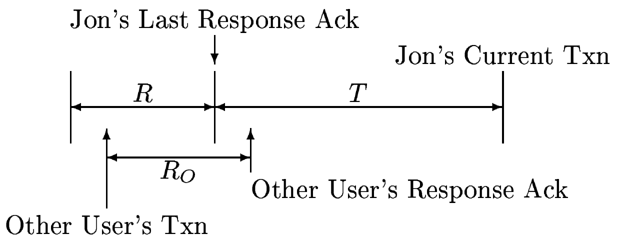

Figure 5: Think Time Greater Than Response Time

will cause the PCB for some user other than Jon to be placed at the head of the chain. A packet arrival during T may be caused either directly (by the arrival of a new transaction from the other user during T) or indirectly (by the arrival of a new transaction from the other user during R). Indirect arrival is illustrated by the lower line in Figure 5: the other user's transaction arrives during interval R, and the corresponding response is sent R time later (denoted by interval R_O in the figure), provoking a transport-layer acknowledgement during T. In either the direct or indirect case, the other user's PCB will precede Jon's when his current transaction arrives. The expected number of PCBs in line ahead of Jon's will be given by $N(T + R)$. Since the probability that the think time will be within an interval of width dT surrounding T is $f(T)$, the corresponding weight (for use in a weighted average yielding the expected number of PCBs preceding Jon's) is just $f(T)N(T + R)$.

On the other hand, if the think time between the transport-level acknowledgement to the response to Jon's last transaction and Jon's current transaction is less than the response time, we have the situation shown in Figure 6. Again, any packet that arrives during the interval T will cause the PCB for some user other than Jon to be placed at the head of the chain. A packet arrival during T may be caused either directly (by the arrival of a new transaction from the other user during T) or indirectly (by the arrival of a new transaction from the other user during T_R). Indirect arrival is illustrated by the lower line in Figure 6: the other user's transaction arrives during interval T_R, the corresponding response is sent R time later (denoted by interval

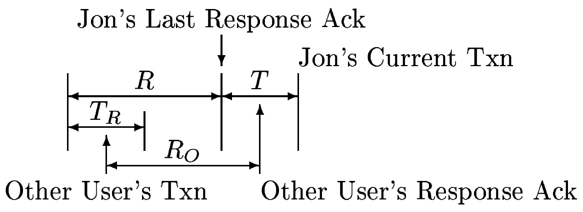

Figure 6: Think Time Less Than Response Time

R_O in the figure), provoking a transport-layer acknowledgement during T. Again, in either the direct or indirect case, the other user's PCB will precede Jon's when his current transaction arrives. The expected number of users in line ahead of Jon will be given by $N(2T)$, resulting in a weight of $f(T)N(2T)$.

Integrating this combined expression from zero to infinity in T gives us the expected number of PCBs preceding Jon's when his transaction entry arrives, given his exponentially-distributed think time (see Equation 5 on the next page). The result for a 200 TPS benchmark is 1,019, 1,045, 1,086, and 1,150 PCBs, corresponding to response times of 0.2, 0.5, 1.0, and 2.0 seconds, respectively (note that 2 seconds is the maximum allowable average response time for the TPC/A benchmark). This performance is somewhat worse than the BSD algorithm's 1,001 PCBs. Note that a TPC/A is not the worst case; if the think times were deterministic (exactly 10 seconds always), Crowcroft's algorithm would look through all 2,000 PCBs on each transaction entry. One example of a system with this behavior is a central server polling its clients, as seen in many point-of-sale terminal applications.

Crowcroft's algorithm does much better during the response-time interval, shown schematically in Figure 7. Any transactions arriving in interval R' will have ac-

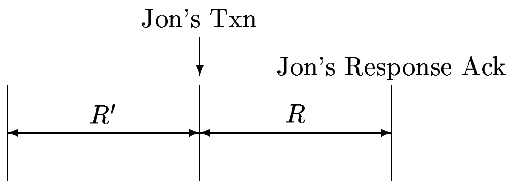

Figure 7: Response Time

knowledgements during interval R, so the number of PCBs preceding Jon's when his acknowledgement arrives will be given by $N(2R)$. The length of the PCB search is 78, 190, 362, and 659 PCBs, for response times of 0.2, 0.5, 1.0, and 2.0 seconds, respectively.

The overall performance of Crowcroft's algorithm will be the average of the performance for the initial transaction entry and the transport-level acknowledgement for the response (see Equation 6 on the next page).

Solving this numerically for 2,000 users gives average search lengths of 549, 618, 724, and 904 PCBs for response times of 0.2, 0.5, 1.0, and 2.0 seconds, respectively. These search lengths represent a significant improvement over the search length of 1,001 resulting from the BSD algorithm.

3.3 Last-Sent/Last-Received Cache

Craig Partridge and Stephen Pink proposed modifying the BSD algorithm so that it caches the PCB corresponding to the last packet sent as well as the last packet received. This modification was motivated by Mogul's work [Mog91], which showed that network traffic often exhibits significant locality. Figure 8 gives a schematic of the PCB list after a packet has been sent on connection "A" and received on connection "B".

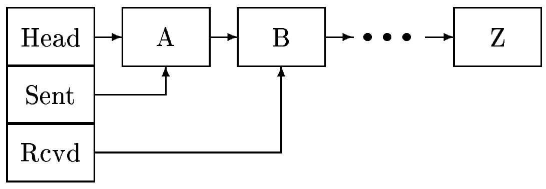

Figure 8: Last-Sent/Last-Received Cache

This modification results in no significant change in the number of PCBs searched for the TCP packet representing the entry of a new transaction, but, like Crowcroft's algorithm, gives a substantial decrease in the number of PCBs that are searched for the transport-level acknowledgement to the TCP packet representing the response. This decrease results in a significant overall reduction in overhead compared to the BSD algorithm in TPC/A benchmarks with a relatively small number of users.

The following analysis assumes that the receive cache is examined before the send cache. The hit rate is a function of the think time T, the response time R, and the network round-trip time D. The analysis derives the number of PCBs searched for three cases: (1) reception of transaction in which $T \geq R+D$ (N_1), (2) reception of transaction in which $T \leq R+D$ (N_2), and (3) reception of transport-level acknowledgement (N_a). The average number of PCBs searched per packet reception is then given by:

$$N = \frac{1}{2}(N_1 + N_2 + N_a) \qquad (7)$$

Note that since N_1 and N_2 are counts of mutually exclusive events, they are simply added together rather than causing a nested average operation.

$$\int_0^R ae^{-aT} \sum_{i=0}^{N-1} i \begin{pmatrix} N-1 \\ i \end{pmatrix} \left(1-e^{-2aT}\right)^i e^{-2aT(N-1-i)} dT \quad + \\ \int_R^\infty ae^{-aT} \sum_{i=0}^{N-1} i \begin{pmatrix} N-1 \\ i \end{pmatrix} \left(1-e^{-a(T+R)}\right)^i e^{-a(T+R)(N-1-i)} dT \tag{5}$$

$$\begin{aligned} C_{\text{Crowcroft}}(N,R) &= \frac{1}{2} \sum_{i=0}^{N-1} i \begin{pmatrix} N-1 \\ i \end{pmatrix} \left(1-e^{-2aR}\right)^i e^{-2aR(N-1-i)} \quad + \\ &\quad \frac{1}{2} \int_0^R ae^{-aT} \sum_{i=0}^{N-1} i \begin{pmatrix} N-1 \\ i \end{pmatrix} \left(1-e^{-2aT}\right)^i e^{-2aT(N-1-i)} dT + \\ &\quad \frac{1}{2} \int_R^\infty ae^{-aT} \sum_{i=0}^{N-1} i \begin{pmatrix} N-1 \\ i \end{pmatrix} \left(1-e^{-a(T+R)}\right)^i e^{-a(T+R)(N-1-i)} dT \end{aligned} \tag{6}$$

3.3.1 Case 1: $T > R + D$

Suppose that one user (call him Stephen) was entering transactions while another user (call him Craig) was trying to enter transactions so as to flush the PCB's corresponding to Stephen's connection from the cache and thus drive his hit ratio down to zero. Figure 9 illustrates one way for Craig to accomplish this. By the time that Stephen enters his next transaction, the response to Craig's transaction has flushed Stephen's PCB from both the send and the receive caches.

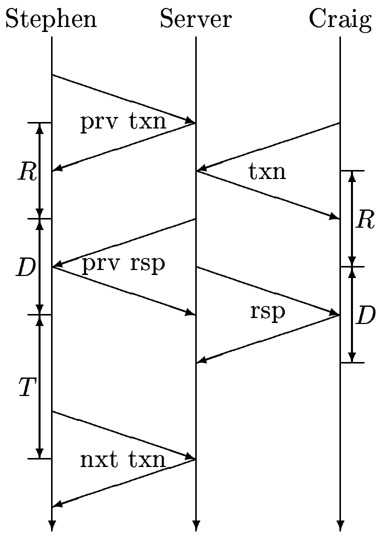

Figure 9: $T > R + D$

Instead, Craig might time his transaction so that his packet arrives at the server after Stephen's response packet has been transmitted but before the transport-level acknowledgement for Stephen's packet has arrived. In this case, Craig's transport-level acknowledgement will flush Stephen's PCB from the send cache. Since $T \geq R + D$, the transport-level acknowledgement to Craig's response will arrive before Stephen's next transaction. This acknowledgement will flush Stephen's PCB from the receive side, forcing a full miss.

Finally, if Craig timed his transaction to arrive at the server after the transport-level acknowledgement to Stephen's response arrived, the transaction and acknowledgement would again completely flush Stephen's PCB from the cache.

In short, if Craig's transaction arrives at the server at any time during the interval of length $T + R + D$ between the arrival of Stephen's two transactions, Craig will succeed in flushing Stephen's PCB from the caches.

The probability of Craig doing this during a TPC/A benchmark is just $1 - e^{-a(T+R+D)}$, so the probability that *none* of the $N - 1$ users other than Stephen will do so is:

$$p_1 = e^{-a(T+R+D)(N-1)} \tag{8}$$

If Stephen's cache survives, only one PCB will need to be examined (both sides of the cache will hold Stephen's PCB). Otherwise, $(N + 5)/2$ PCBs must be searched: the two cached PCBs and $(N + 1)/2$ of the PCBs in the chain, on the average. Thus, the expected number of PCBs to be searched for a given value of the think time T will be as shown in Equation 9.

$$\begin{aligned} N(T) &= e^{-a(T+R+D)(N-1)} + \\ &\quad \left(1-e^{-a(T+R+D)(N-1)}\right) \frac{N+5}{2} \end{aligned} \tag{9}$$

As noted in Section 3.2, the probability that the TPC/A think time will be in an interval of size dT centered on some value T is $ae^{-aT}dT$, and integrating this quantity

$$N_1 = \int_{R+D}^{\infty} ae^{-aT} \left(e^{-a(T+R+D)(N-1)} + \left(1 - e^{-a(T+R+D)(N-1)}\right) \frac{N+5}{2} \right) dT \tag{10}$$

multiplied by Equation 9 over the interval of interest gives the expected number of PCBs searched (see Equation 10 on the next page). Integrating and simplifying yields:

$$N_1 = \frac{N+5}{2} e^{-a(R+D)} - \frac{N+3}{2N} e^{-a(R+D)(2N-1)} \tag{11}$$

This equation may seem strange at first, because the number of PCBs searched *decreases* with increasing response time and round-trip delay. The reason for this is that T has been constrained to be greater than $R + D$ for this case. The exponential distribution for T means that larger values of T are less likely to occur; this overwhelms the increased miss rate caused by large values of $R + D$.

3.3.2 Case 2: $T \leq R + D$

If Stephen's think time is not greater than the response time plus the round-trip delay, then his PCB cache can survive an intervening transaction. Figure 10 shows how this can happen: the receive-side cache is still live when Stephen's next transaction arrives.

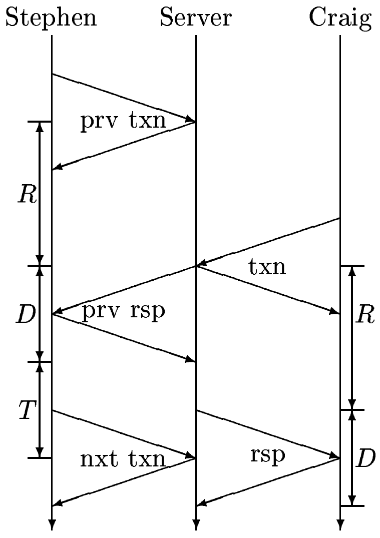

Figure 10: $T \leq R + D$

In order to fully flush Stephen's PCB from the cache, Craig must cause a packet to arrive at the server during the interval of duration T between the arrival of the transport-level acknowledgement for Stephen's last response and the arrival of Stephen's next transaction. Craig can arrange this by having his transaction arrive either during this interval or during the interval of duration T that begins $R+D$ time units before this interval.

The probability of Craig doing this during a TPC/A benchmark is just $1 - e^{-2aT}$, so the probability that none of the $N - 1$ users other than Stephen will do so is:

$$p_2 = e^{-2aT(N-1)} \tag{12}$$

If the receive-side cache is examined first,[5] then the miss penalties are exactly the same as in Case 1. Thus, the above expression is combined with the TPC/A think time and integrated over the interval of interest (see Equation 13 on the next page). This equation may be integrated and simplified as shown in Equation 14.

$$\begin{aligned} N_2 &= \frac{N+5}{2}\left(1 - e^{-a(R+D)}\right) - \\ &\quad \frac{N+3}{2(2N-1)}\left(1 - e^{-a(R+D)(2N-1)}\right) \end{aligned} \tag{14}$$

As the response time, round-trip delay, and number of users increase, the number of PCBs searched approaches $(N + 5)/2$. In other words, as the stress on the cache increases, the performance converges to that of an uncached linked list plus the overhead imposed by the cache.

3.3.3 Case 3: Acknowledgements

Suppose that Craig is trying to flush Stephen's PCB from the cache so that the transport-level acknowledgement to Stephen's response must search the full PCB list. One way for Craig to accomplish this is illustrated in Figure 11. Here, Craig has timed his transaction so

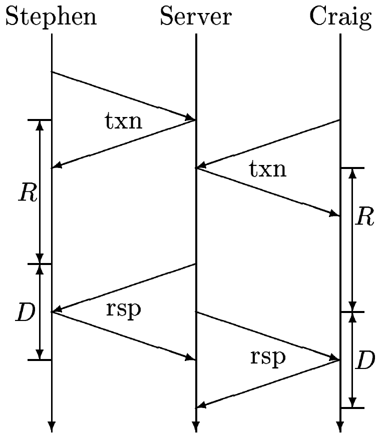

Figure 11: Transport-Level Acknowledgements

[5]Examining the receive-side cache makes most sense for TCP data packets, while examining the send-side cache first makes most sense for TCP acknowledgement packets.

$$N_2 = \int_0^{R+D} ae^{-aT}\left(e^{-2aT(N-1)} + \left(1 - e^{-2aT(N-1)}\right)\frac{N+5}{2}\right) dT \qquad (13)$$

that his response packet will flush Stephen's PCB from the send-side cache (Craig's transaction packet will already have flushed Stephen's PCB from the receive-side cache). Craig could also time his transaction to arrive during the interval of length D between the transmission of Stephen's response packet and the reception of the corresponding transport-level acknowledgement.

Thus, Craig has two windows of duration D in which to enter his transaction. The probability of a normal TPC/A user hitting either or both of these windows is:

$$p_a = e^{-2aD} \qquad (15)$$

Assuming that the send-side cache is examined first when processing an acknowledgment, the hit and miss penalties are identical to those in Cases 1 and 2. Since D is assumed constant, no integration is required, and the expected number of PCBs searched is simply:

$$N_a = \frac{N+5}{2} - \frac{N+3}{2} e^{-2aD(N-1)} \qquad (16)$$

As D and N increase, this expression approaches $\frac{N+5}{2}$, as expected. As D decreases toward zero or N decreases toward one, the expression approaches just one (the number of accesses required to check the send side of the cache).

3.3.4 Overall Result

Substituting the expressions for N_1, N_2, and N_a into Equation 7 and simplifying yields the result shown in Equation 17 (which is shown on the following page). This expression approaches just $\frac{N+5}{2}$ as N increases, as expected.

Solving this numerically for 2,000 users and round-trip delays of 1, 10, and 100 milliseconds gives average search lengths of 667, 993, and 1002 PCBs, respectively. The algorithm is extremely insensitive to the value of R for large values of N. For short round-trip delays, this is significantly better than the BSD algorithm, and is roughly of the order of Crowcroft's algorithm.

3.4 Sequent

Sequent's algorithm maintains a simple linear list for each of several hash chains, each containing a single-entry cache containing the PCB last found on that hash chain.[6] Figure 12 shows a schematic of this data structure just after the arrival of packets on connections "A0" and "Bn". Note that hash chain 1 is empty, and thus its cache points nowhere.

[6]A similar approach was suggested on the tcp-ip mailing list by Lance Vissner [Vis91].

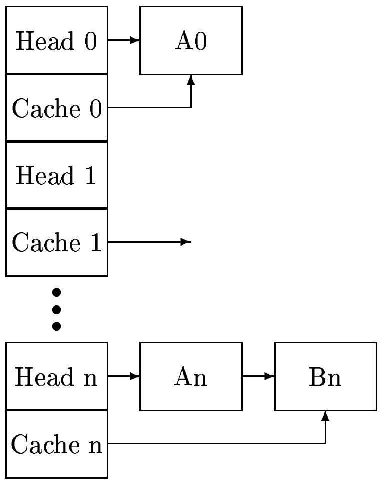

Figure 12: Sequent PCB List

The hit rate for the PCB cache is H/N where H is the number of hash chains. This rate comes to just over 0.95% given the installation default of 19 hash chains running a 200 TPC/A TPS benchmark. The worst-case cost of a miss is a linear search scanning N/H PCBs, 106 for the installation default number of hash chains. The average number of PCBs that must be examined is just one if we hit the cache and an additional $(N/H+1)/2$ if we miss. The probability of a hit is just H/N, and the probability of a miss is $(N-H)/N$. Thus it is tempting to assume:

$$\begin{aligned} C_{\mathrm{SQNT}}(N,H) &= 1 + \frac{(\frac{N}{H})^2 - 1}{2\frac{N}{H}} && (18)\\ &= C_{\mathrm{BSD}}(\frac{N}{H}) \ , && (19)\end{aligned}$$

approaching $N/2H$ for large N.

However, the decreased number of PCBs serviced by each cache greatly increases the probability that there will be no packets arriving at the server during a given transaction's response-time interval. The probability that no packets will arrive during the response-time interval is shown as:

$$p = e^{-2aR(\frac{N}{H}-1)} \qquad (20)$$

in which a is 0.1 seconds per transaction for the TPC/A benchmark, R is the response-time interval, N is the number of TPC/A users, and H is the number of hash chains. This probability is about 1.5% for a 2000-user benchmark with a 200-millisecond response time and 19 hash chains. Decreasing the number of users or the

$$N = \frac{N+3}{4N(2N-1)}\left((1-N)e^{-a(R+D)(2N-1)} - N\right) + \frac{N+5}{2} - \frac{N+3}{4}e^{-2aD(N-1)} \qquad (17)$$

response time or increasing the number of hash chains will greatly increase this probability. For example, if the number of hash chains is increased to 51, the probability increases to almost 21%. These compare quite favorably to the 1.9×10^{-35} probability for the single-chain BSD algorithm.

If no packets arrive during the response-time interval, only the single cached PCB need be examined. Otherwise, $(N/H+1)/2$ PCBs will be examined on the average. The transport-level acknowledgment packet must thus search

$$e^{-2aR(\frac{N}{H}-1)} + \left(1 - e^{-2aR(\frac{N}{H}-1)}\right)\frac{\frac{N}{H}+1}{2} \qquad (21)$$

PCBs on the average. Assuming negligible loss rates, half of the packets will be acknowledgements, so the overall expected number of PCBs to search is given by the mean of Equations 19 and 21, as shown in Equation 22 on the following page.

This equation yields an average cost of a linear scan of 53.0 PCBs for a 200 TPC/A TPS benchmark with 19 hash chains and a 200-millisecond response time. In contrast, Equation 19 predicts 53.6 for a little more than 1% error. The error gets larger with smaller numbers of users, smaller response times, and larger numbers of hash chains, exceeding 10% if 51 hash chains are substituted into the previous example.

Either equation predicts an order of magnitude improvement over the BSD algorithm, Crowcroft's proposed algorithm, or Partridge's and Pink's proposed algorithm, and is more amenable to the sizes of current on-chip data caches. Of course, the system administrator may increase the value of H in order to get even better performance, at the expense of a small increase in the memory used for the hash chain headers.

Although hit ratios of a few percent are typical for a TPC/A run, ratios as high as 30% have been observed. However, these runs were done using old versions of database software that sent three times as many packets for each transaction as necessary. In fact, if all these extra packets arrived simultaneously, the hit rate would be as high as 67%. Nonetheless, the number of PCBs searched *per transaction* is at least as large as that for software exhibiting "poor" hit ratios due to more efficient use of network resources. Focusing strictly on hit ratio is a common pitfall. The hit ratio is only part of the story; this is just one example where the miss penalty dominates the hit ratio.

3.5 Comparison

Figure 13 plots the cost of the PCB search against the number of TPC/A users. The lines labelled "MTF 1.0", "MTF 0.5", and "MTF 0.2" show the performance of Crowcroft's move-to-front algorithm given a response time of 1.0, 0.5, and 0.2 seconds, respectively. The line labelled "SR 1" shows the performance of Partridge's and Pink's send-receive cache given a round-trip time of 1 millisecond. Crowcroft's move-to-front

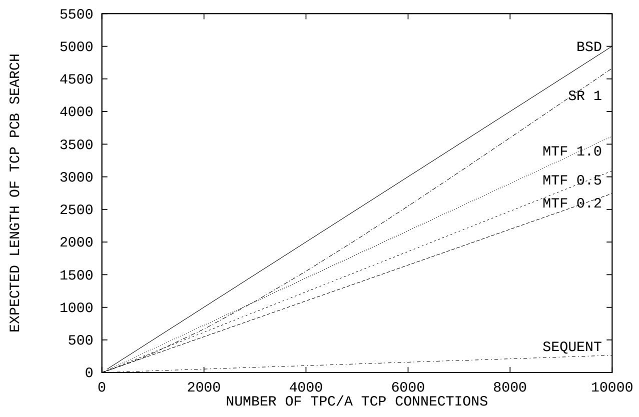

Figure 13: Comparison of TCP Demultiplexing Algorithms

algorithm is significantly better than the stock BSD algorithm, and improves as the response time decreases. Partridge's and Pink's send/receive cache algorithm is significantly better than the stock BSD algorithm for small numbers of users (see Figure 14), and asymtotically approaches the BSD algorithm's performance for large numbers of users. This behavior is due to the fact that the send/receive cache relies on packet trains, which rarely occur in large TPC/A benchmarks.

The Sequent algorithm is roughly an order of magnitude better than the other algorithms. The only added cost of the Sequent algorithm over BSD is the memory required for the hash-chain headers and the computation of the hash function itself. Memory is still decreasing rapidly in price, and efficient hash functions for protocol addresses are well known [Jai89, McK91].

One could imagine combining move-to-front with hash chains. However, better results can be obtained simply by increasing the number of hash chains. For example, if the number of hash chains in the above example is increased from 19 to 100, the average number of PCBs searched drops from 53 to less than 9. This factor-of-five improvement compares favorably with the best-

$$C_{\mathrm{SQNT}}(N,H,R)=\frac{1}{2}\left[\frac{1-\frac{N}{H}}{2}e^{-2aR(\frac{N}{H}-1)}+\frac{2\frac{N}{H}^2+3\frac{N}{H}-1}{2\frac{N}{H}}\right] \tag{22}$$

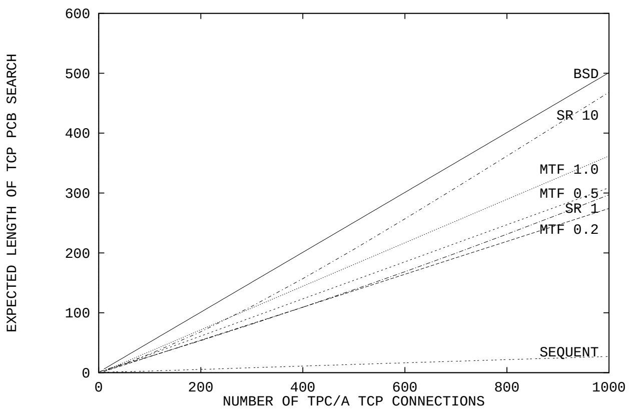

Figure 14: Comparison of TCP Demultiplexing Algorithms (Detail)

case factor-of-two improvement that would be obtained by adding move-to-front. Since a relatively small number of hash chains can reduce the PCB-lookup overhead to an insignificant fraction of the other packet-reception overheads,[7] there is little motivation to combine move-to-front.

In addition, this reduction in PCB-searching reduces the need to add connection IDs to TCP, such as those found in TP4, X.25, and XTP. These protocols allow the two communicating hosts to negotiate the value of a pair of small integers, called connection IDs, in each data packet header. These connection IDs are typically used to directly index an array of PCBs, thus completely eliminating the need to search. The much cheaper search provided by hashing eliminates the motivation for connection IDs on hosts that must do significant per-packet processing such as that required by TPC/A.

4 Conclusions

Good analytic approximations are available to describe the behavior of various TCP demultiplexing algorithms when presented TPC/A-style traffic. These approximations show that heads-down data-entry applications (and benchmarks based on them) result in very poor performance from the BSD TCP demultiplexing algorithm. Significant improvement can be obtained through use of Jon Crowcroft's move-to-front modification to this algorithm and Craig Partridge's and Stephen Pink's last-sender/last-receiver scheme, but order-of-magnitude improvements result from application of hashing techniques such as those used in the Sequent TCP product. These approximations have been qualitatively confirmed by benchmarks.

The Sequent algorithm also greatly reduces the need to add new features to the protocol itself (such as connection IDs) that would eliminate the need to search for PCBs. In fact, it is far from clear whether such an improvement would win widespread acceptance unless combined with significant new capabilities. One example might be features allowing applications to specify their bandwidth and delay requirements, which may be necessary for multimedia applications.

[7]Note that the number of users is usually sharply limited by other factors such as memory size, swap space, and disk bandwidth. If the system could really support an infinite number of users, this argument would not be valid.

References

[Cro91] Jon Crowcroft. Re: Inefficient demultiplexing by 4.3 TCP/IP. Message-ID 2142@ucl-cs.uucp to tcp-ip list, December 1991.

[Dov90] Ken F. Dove. A high capacity TCP/IP in parallel STREAMS. In *UKUUG Conference Proceedings*, London, June 1990.

[Fel90] David C. Feldmeier. Multiplexing issues in communication system design. In *SIGCOMM '90 Symposium*, pages 209–219, Philadelphia, PA, September 1990.

[Gar90] Arun Garg. Parallel STREAMS. In *USENIX Conference Proceedings*, Berkeley CA, February 1990.

[Gra91] Jim Gray. *The Benchmark Handbook for Database and Transaction Processing Systems.* Morgan Kaufmann, 1991.

[HJ91] John L. Hennessy and Norman P. Jouppi. Computer technology and architecture: An evolving interaction. *IEEE Computer*, pages 18–28, September 1991.

[HL86] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research.* Holden-Day, 1986.

[Jac88] Van Jacobson. Congestion avoidance and control. In *SIGCOMM '88*, pages 314–329, August 1988.

[Jai89] Raj Jain. A comparison of hashing schemes for address lookup in computer networks. Technical Report DEC-TR-593, Digital Equipment Corporation, February 1989.

[JR86] Raj Jain and Shawn Routhier. Packet trains–measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, SAC-4(6):986–995, September 1986.

[McK91] Paul E. McKenney. Stochastic fairness queuing. *Internetworking: Theory and Experience*, 2:113–131, 1991.

[Mog91] Jeffrey C. Mogul. Network locality at the scale of processes. In *Proceeding of SIGCOMM '91*, Zurich, September 1991.

[Pos81] J.B. Postel. Transmission Control Protocol. Technical Report RFC793, Network Information Center, SRI International, September 1981.

[PP91] Craig Partridge and Stephen Pink. A faster UDP. Swedish Institute of Computer Science Tecnical Report, August 1991.

[SC91] Harold S. Stone and John Cocke. Computer architecture in the 1990s. *IEEE Computer*, pages 30–38, September 1991.

[Vis91] Lance Vissner. Re: Inefficient demultiplexing by 4.3 TCP/IP. Message-ID visser.691884939@convex.convex.com to tcp-ip list, December 1991.