Efficient Demultiplexing of Incoming TCP Packets

Paul E. McKenney and Ken F. Dove
Sequent Computer Systems, Inc.

ABSTRACT: Many recent protocol optimizations put forth for the TCP/IP protocol suite assume that a large component of the traffic is bulk-data transfers, which result in packet trains.[1] These packet trains favor simple caching algorithms such as those found in the BSD implementation of TCP.

Although BSD's algorithms have been very successful in many situations, applications that do not form packet trains can cause these algorithms to perform very poorly. 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; this use is typically characterized by large numbers of connections,[2] low per-connection packet rate, large aggregate packet rate, and small packets. This combination of characteristics results in an extremely low incidence of packet trains.

1. A packet train is a contiguous group of packets with the same destination; a packet is a block of data that is transmitted through a computer communications network as a unit.
2. For the purposes of this paper, a computer network connection can be thought of as a pathway that carries data between a specific user and his application.

This paper demonstrates that this causes BSD's caching optimizations to be ineffective. It will analyze some alternative optimizations, one of which performs more than an order of magnitude better than BSD's for OLTP applications, with almost no performance penalty for other applications.

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. It does so by mapping the packet's source and destination Internet Protocol (IP) addresses and ports[3] to the proper PCB. Since the IP addresses and ports are 96 bits long, simple indexing schemes are not feasible; more complex schemes must instead be used.

BSD originally used a simple linear linked list of PCBs. Sequent's initial implementation of TCP followed BSD closely, retaining the simple 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 via local-area networks. This resulted in, among other things, the algorithm described in Section 3.3.

At the same time, the first-generation TCP was causing performance problems in sales benchmarks. Substituting the new algorithm resulted in throughput increases of up to a factor of three.

Also about this time, Van Jacobson was conducting research aimed at increasing TCP's single stream performance. As a result of this re-

3. An IP address identifies the network interface through which the packet is to be sent or received; a port identifies the application that caused the packet to be sent or is to receive the packet. Taken together, the IP addresses and ports uniquely identify a TCP connection.

search, 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.

Later, the Transaction Processing Council published the TPC/A online-transaction-processing benchmark [Gra91], which quickly came into common use by database software and platform vendors. This is a very important development, as 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 analytic comparisons of the effects of different protocols and algorithms.

This article shows that the TPC/A benchmark is almost free of packet trains, thereby causing the BSD algorithm to perform very poorly. It also presents analysis predicting that an alternative algorithm suggested by Jon Crowcroft [Cro91] should achieve significantly higher performance, and finally presents an alternative algorithm that achieves an additional order of magnitude improvement for the TPC/A benchmark while maintaining good performance in other situations. This increase in performance has protocol-design implications, as 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 BSD algorithm, John Crowcroft's algorithm, and Sequent's algorithm; Section 4 presents experimental results; and Section 5 presents conclusions.

2. *TPC/A Benchmark*

The TPC/A benchmark simulates a banking system in which simulated customers make randomly-generated deposits to and withdrawals from a simulated 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.

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

3. Analysis

The preceding 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.[4]

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

4. Although delayed acknowledgements 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.

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.

Since the negative exponential distribution is memoryless, each of the 2,000 users is 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]); it 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.[5]

The following sections analyze the average cost of the BSD algorithm, the "move to front" algorithm proposed by Jon Crowcroft, 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".

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 1,000 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:

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

5. This 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.

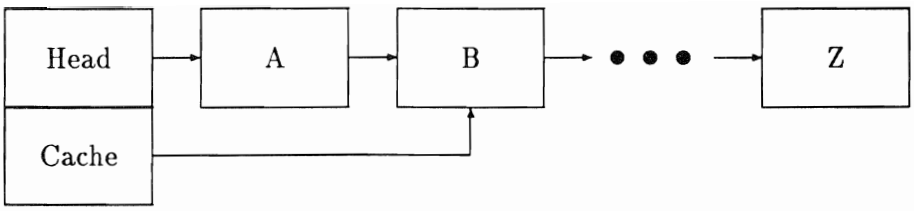

Figure 1: BSD PCB List After Arrival of Packet on Connection B

approaching $N/2$ for large N. This 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. Since the data contained in all 2,000 PCBs will not fit into on-chip data caches for any currently-available microprocessor that we are aware of, 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.[6] 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.

6. 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.

3.2 *"Move to front" List*

Jon Crowcroft proposed maintaining a linear list with a "move to front" heuristic; 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.

This heuristic results in a slight increase in the number of PCBs searched for the TCP packet representing the entry of a new transaction but a substantial decrease in the number of PCBs searched for the transport-level acknowledgement to the TCP packet representing the response. This 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 is just the cumulative distribution function for the exponential distribution. The expected number of users from a total of

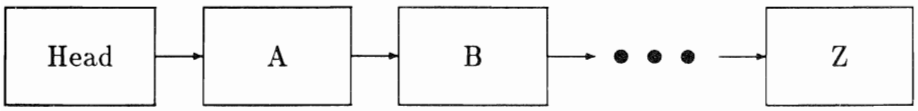

Figure 2: Jon's PCB List Before Arrival of Packet on Connection B

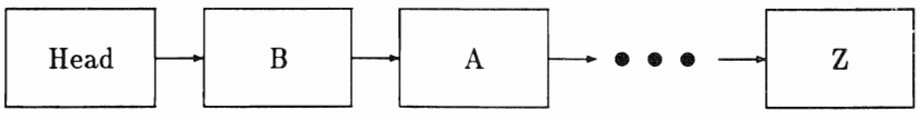

Figure 3: Jon's PCB List After Arrival of Packet on Connection B

$(N - 1)$ users (all of them but Jon) to enter at least one transaction during this time will be

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

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, $(1 - e^{-aT})^i$ 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" 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.

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

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

The approximation gets better as dT gets smaller. This is just the dis-

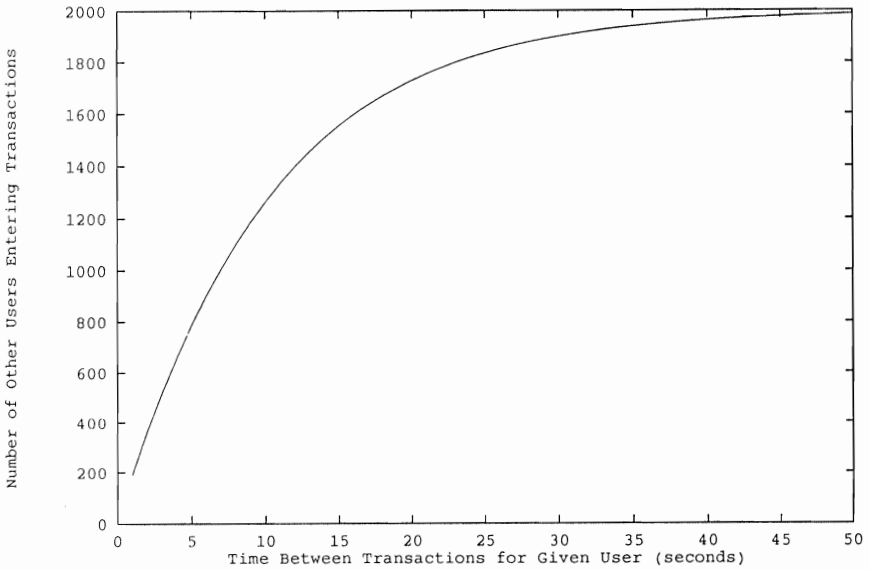

Figure 4: $N(T)$ for 2,000 TPC/A Users

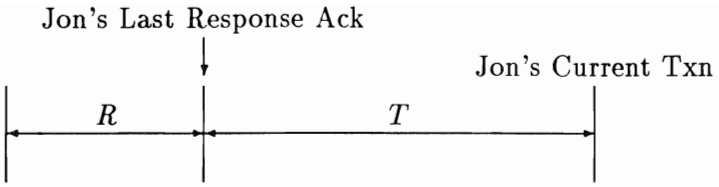

Figure 5: Think Time Greater Than Response Time

tribution 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 transaction arriving during either interval R or T will result in a transaction or acknowledgement packet, respectively, arriving in interval T. Therefore the corresponding 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. Here, any transaction that arrives during the interval labelled T_R will have a transport-level acknowledgement that arrives during the interval labelled T. Any transaction arriving during either the T_R or T intervals will result in a transaction or acknowledgement packet, respectively, arriving during interval T. The corresponding PCB will precede Jon's when his current transaction arrives. The

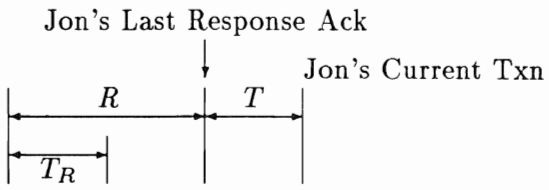

Figure 6: Think Time Less Than Response Time

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:

$$\int_0^R ae^{-aT} \sum_{i=0}^{N-1} i\binom{N-1}{i}(1-e^{-2aT})^i e^{-2aT(N-1-i)}dT + \int_R^\infty ae^{-aT} \sum_{i=0}^{N-1} i\binom{N-1}{i}(1-e^{-a(T+R)})^i e^{-a(T+R)(N-1-i)}dT. \quad (5)$$

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 is somewhat worse performance 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), Jon'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.

Jon's algorithm does much better during the response-time interval, shown schematically in Figure 7. Any transactions arriving in interval R' will have acknowledgments during interval R, so the number of PCBs preceding Jon's when his acknowledgment 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 Jon's algorithm will be the average of the performance for the initial transaction entry and the transport-level acknowledgement for the response. This is given by:

$$C_{\text{Jon}}(N, R) = \frac{1}{2}\sum_{i=0}^{N-1} i\binom{N-1}{i}(1-e^{-2aR})^i e^{-2aR(N-1-i)} + \frac{1}{2}\int_0^R ae^{-aT} \sum_{i=0}^{N-1} i\binom{N-1}{i}(1-e^{-2aT})^i e^{-2aT(N-1-i)}dT + \frac{1}{2}\int_R^\infty ae^{-aT} \sum_{i=0}^{N-1} i\binom{N-1}{i}(1-e^{-a(T+R)})^i e^{-a(T+R)(N-1-i)}dT \quad (6)$$

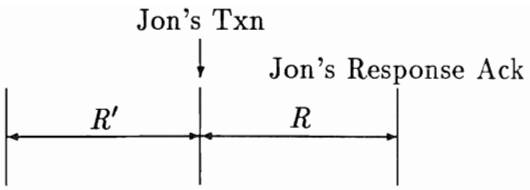

Figure 7: Response Time

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. This is a significant improvement over the search length of 1,001 resulting from the BSD algorithm.

3.3 Sequent

Sequent's algorithm, designed and implemented by Ken Dove [Dov90], 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.[7] Figure 8 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.

The hit rate for the PCB cache is H/N where H is the number of hash chains. This comes to just over 0.95% given the installation default of 19 hash chains running a 200 TPC/A TPS benchmark. The average 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:

$$C_{\text{SQNT}}(N, H) = 1 + \frac{\left(\frac{N}{H}\right)^2 - 1}{2\frac{N}{H}} \tag{7}$$

$$= C_{\text{BSD}}\left(\frac{N}{H}\right), \tag{8}$$

approaching $N/2H$ for large N.

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

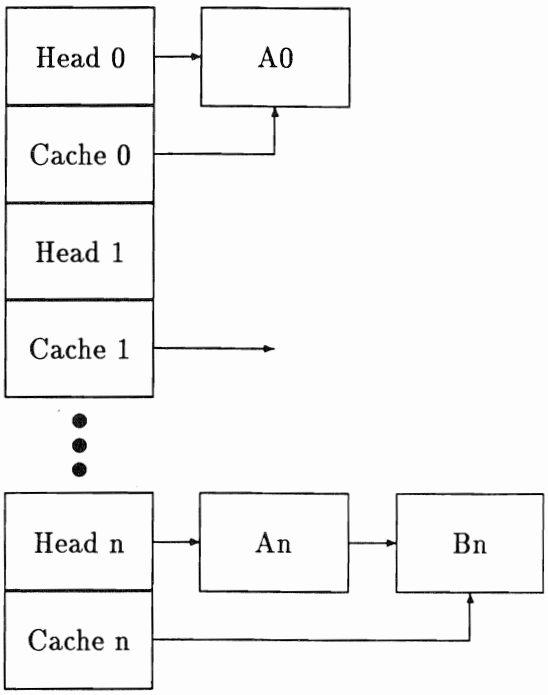

Figure 8: Sequent PCB List

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 given by:

$$p = e^{-2aR(\frac{N}{H}-1)} \tag{9}$$

where 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 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 acknowledgement packet must thus search

$$e^{-2aR(\frac{N}{H}-1)} + (1 - e^{-2aR(\frac{N}{H}-1)})\frac{\frac{N}{H} + 1}{2} \tag{10}$$

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 8 and 10:

$$C_{\text{SQNT}}(N, H, R) = \frac{1}{2}\left[e^{-2aR(\frac{N}{H}-1)}\frac{1 - \frac{N}{H}}{2} + \frac{2\frac{N^2}{H} + 3\frac{N}{H} - 1}{2\frac{N}{H}} \right] \tag{11}$$

This 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 8 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 case is an order of magnitude improvement over the BSD algorithm or Jon'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 that exhibits "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.

3.4 Comparison

Figure 9 plots the cost of the PCB search against the number of TPC/A users. Jon's "move to front" algorithm is significantly better than the stock BSD algorithm, and improves as the response time decreases. The Sequent algorithm is roughly an order of magnitude better than either of 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 Internet 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-case factor-of-two improvement that would be obtained by adding move-to-front.

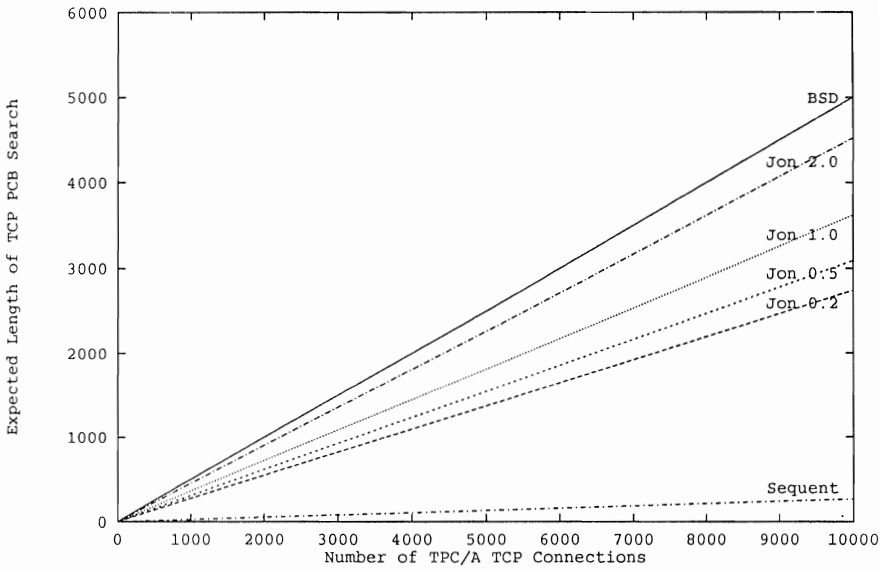

Figure 9: Comparison of TCP Demultiplexing Algorithms

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 *Measurements*

Section 1 described how a particular application benefitted dramatically, though subjectively, from the introduction of Sequent's algorithm.

Although TPC/A forms a good basis for an analytic comparison of protocol algorithms, a real TPC/A run requires additional heavyweight components such as databases and transaction monitors. These components bring with them a large number of tuning parameters, some of whose settings are fairly sensitive to available CPU power, which, as we have seen, is significantly affected by the choice of TCP demultiplexing algorithm. This makes it impossible to perform a simple experiment to fairly compare these algorithms—the optimal TPS for one algorithm would be obtained using quite different tuning values than those that yielded optimal TPS for another algorithm.

We therefore used a simple TCP connection-setup experiment to demonstrate the difference between the BSD and Sequent algorithms.[8] This experiment uses a client machine "C" and a server machine "S" connected via Ethernet. Both machines used old, relatively slow hardware.

In the first run, both "C" and "S" use the BSD algorithm. "C" initiates 3,600 TCP connections to "S" as nearly simultaneously as possible, then transmits five 100-byte datagrams on each connection as rapidly as possible. A total of 46,800 packets are transmitted counting data packets and protocol-overhead packets but neglecting

8. The analysis presented earlier in the paper convinced us not to implement Jon's algorithm.

retransmissions. Each connection transmits as soon as it is fully set up, so that data transmissions from those lucky connections that get set up quickly interfere with the connection setup of the not-so-lucky connections that get set up later. The figure of merit is the time from the transmission of the first connection setup (TCP SYN packet) from the client to the reception of the last connection completion (TCP SYN-ACK packet) at the client.

The second run is identical, except that both "C" and "S" use the Sequent algorithm with 19 hash lines and that "C" initiates 5,040 TCP connections, about 40% more than in the first run. A total of 65,520 packets will be transmitted, again counting data and protocol packets but neglecting acknowledgements.

The first run requires 906 seconds to set up 3600 connections, while second run requires only 193 seconds to set up 5040 connections. This is due to the fact that the first run had to examine at least 84 million PCBs, while the second run only had to examine about 8.7 million PCBs.

The overhead of TCP demultiplexing can have a dramatic effect on overall system performance.

5 *Conclusions*

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, but order-of-magnitude improvements result from application of hashing techniques such as those used in the Sequent TCP product.

These improvements also greatly reduce 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 that 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.

References

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

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

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

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

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

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

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

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

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

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

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

[submitted Jan. 17, 1992; revised Apr. 3, 1992; accepted Apr. 10, 1992]