Predictable Communication on Unpredictable Networks: Implementing BSP over TCP/IP

Stephen R. Donaldson[1], Jonathan M.D. Hill[1], and David B. Skillicorn[2]

[1] Oxford University Computing Laboratory, UK.
[2] CISC, Queen's University, Canada

Abstract. The BSP cost model measures the cost of communication using a single architectural parameter, g, which measures permeability of the network to continuous traffic. Architectures, typically networks of workstations, pose particular problems for high-performance communication because it is hard to achieve high throughput, and even harder to do so predictably. Yet both of these are required for BSP to be effective. We present a technique for controlling applied communication load that achieves both. Traffic is presented to the communication network at a rate chosen to maximise throughput and minimise its variance. Performance improvements as large as a factor of two over MPI can be achieved.

1 Introduction

The BSP (Bulk Synchronous Parallel) model [10, 8] views a parallel machine as a set of processor-memory pairs, with a global communication network and a mechanism for synchronising all processors. A BSP calculation consists of a sequence of *supersteps*. Each superstep involves all of the processors and consists of three phases: (1) processor-memory pairs perform a number of computations on data held locally at the start of a superstep; (2) processors communicate data into other processor's memories; and (3) all processors synchronise.

The BSP cost model treats communication as an aggregate operation of the entire executing architecture, and models the cost of delivery using a single architectural parameter, the permeability, g. This parameter can be intuitively understood as defining the time taken for a processor to communicate a single word to a remote processor, *in the steady state where all processors are simultaneously communicating*. The value of the g parameter will depend upon: (1) the bisection bandwidth of the communication network topology; (2) the protocols used to interface with and within the communication network; (3) buffer management by both the processors and the communication network; and (4) the routing strategy used in the communication network.

However, the BSP runtime system also makes an important contribution to the performance by acting to improve the effective value of g by the way it uses the architectural facilities. For example, [6] shows how orders of magnitude

improvements in g can be obtained, for architectures using point-to-point connections, by packing messages before transmission, and by altering the order of transmission to avoid contention at receivers.

In this paper, we address the problem raised by shared-media networks and protocols such as TCP/IP, where there is far greater potential to waste bandwidth. For example, if two processors try to send more or less simultaneously, collision in the ether means that neither succeeds, and transmission capacity is permanently lost. The problem is compounded because it is hard for each processor to learn anything of the global state of the network. Nevertheless, as we shall show, significant performance improvements are possible.

We describe techniques, specific to our implementation of *BSPlib* [5], that ensure that the variation in g is minimised for programs running over bus-based Ethernet networks. Compared to alternative communications libraries such as Argonne's implementation of MPI [2], these techniques have an absolute performance improvement over MPI in terms of the mean communication throughput, but also have a considerably smaller standard deviation. Good performance over such networks is of practical importance because networks of workstations are increasingly used as practical parallel computers.

2 Minimising g in Bus-Based Ethernet Networks

Ethernet (IEEE 802.3) is a bus-based protocol in which the media access protocol, 1-persistent CSMA/CD (Carrier Sense Multiple Access with Collision Detection) proceeds as follows. A station wishing to send a frame listens to the medium for transmission activity by another station. If no activity is sensed, the station begins transmission and continues to listen on the channel for a collision. After twice the propagation delay, 2τ, of the medium, no collision can occur, as all stations sensing the medium would detect that it is in use and will not send data. However, a collision may occur during the 2τ window. On detection, the transmitting station broadcasts a jamming signal onto the network to ensure that all stations are notified of the collision. The station recovers from a collision by using a *binary exponential back-off* algorithm that re-attempts the transmission after $t \times 2\tau$, where t is a random variable chosen uniformly from the interval $[0, 2^k]$ (where k is the number of collisions this attempted transmission has experienced). For Ethernet, the protocol allows k to reach ten, and then allows another six attempts at $k = 10$ (see, for example, King [7]).

Analysis of this protocol (Tasaka [9]) shows that $S \to 0$ as $G \to \infty$ (where S is the rate of successful transmissions and G the rate at which messages are presented for delivery), whereas for a p-processor BSP computer one would expect that $S \to B$ as $G \to \infty$, from which one could conclude that $g = p/B$; where B is a measure of the bandwidth.

In the case of BSP computation over Ethernet, the effect of the exponential backoff is exaggerated (larger delays for the same amount of traffic) because the access to the medium is often synchronised by the previous barrier synchronisation and the subsequent computation phase. For perfectly-balanced

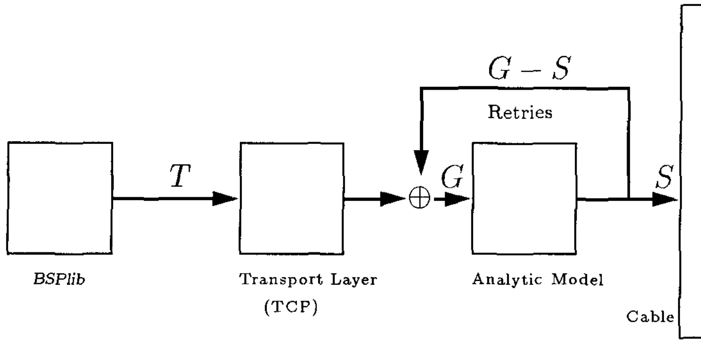


Fig. 1. Schematic of aggregated protocol layers and associated applied loads

computations and barrier synchronisations, all processors attempt to get their first message onto the Ethernet at the same time. All fail and back off. In the first phase of the exponential back-off algorithm, each of p processors choose a uniformly-distributed wait period in the interval $[0, 4\tau]$. Thus the expected number of processors attempting the retransmits in the interval $[0, 2\tau]$ is $p/2$, making secondary collisions very likely. If the processors are not perfectly balanced, and a processor gains access to the medium after a short contention period, then that process will hold the medium for the transmission of the packet, which will take just over $1000\mu s$ for 10Mbps Ethernet. With high probability, many of the other processors will be synchronised by this successful transmission due to the 1-persistence of this protocol. The remaining processors will then contend as in the perfectly-balanced scenario.

In terms of the performance model, this corresponds to a high applied load, G, albeit for a short interval of time. If S were (at least) linear in G then this burstiness of the applied load would not be detrimental to the throughput and would average out.

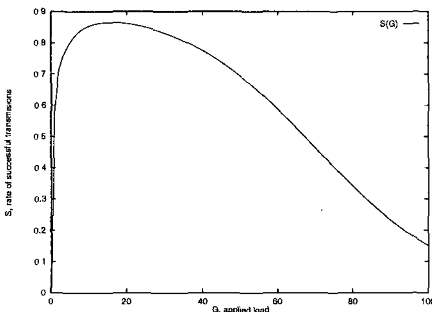


Fig. 2. Plot of applied load (G) against successful transmissions (S)

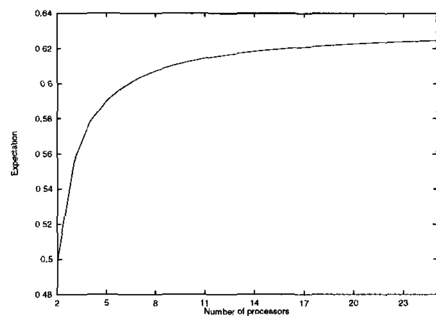


Fig. 3. Contention expectation for a particular slot as a function of p

Fortunately, the BSP model allows assumptions to be made at the global level based on local data presented for transmission. At the end of a superstep and before any user data is communicated, *BSPlib* performs a reduction, in which all processors determine the amount of communication that each processor intends sending. From this, the number of processors involved in the communication is determined. For the rest of the communication the number of processors involved and the amount of data is used to regulate (at the transport level) the rate at which data is presented for transmission on the Ethernet. By using BSD Socket options (TCP_NDELAY), the data presented at the transport layer are delivered immediately to the MAC layer (ignoring the depth of the protocol stack and the availability of a suitable window size). Thus, by pacing the transport layer, pacing can be achieved at the MAC or link layer. This has the effect of removing burstiness from the applied load.

Most performance analyses give a model of random-access broadcast networks which provide an analytic, often approximate, result for the successful traffic, S, in terms of the offered load, G. Hammond and O'Reilly [3] present a model for slotted 1-persistent CSMA/CD in which the successful traffic, S, (or efficiency achieved) can be determined in terms of the offered load G, the end-to-end propagation delay, τ (bounded by $25.65\mu s$, i.e., the time taken for a signal to propagate $2500m$ of cable and 4 repeaters), and E, the frame transmit time (which for 10Mbps Ethernet with a maximum frame size of 1500 bytes is approximately $1200\mu s$). Figure 2 shows the predicted rate of successful traffic against applied load, assuming that that jamming time is equal to τ.

Since both S, the rate of successful transmissions, and G are normalised with respect to E, S is also the channel efficiency achieved on the cable. T, shown in Figure 1, also normalised with respect to E, is the load applied by *BSPlib* on the transport layer. Our objective is to pace the injection of messages into the transport layer such that T, on average, is equal to a steady-state value of S without much variance. The value of T determines the position on the S–G curve of Figure 2 in a *steady state*; in particular, T can be chosen to maximise S. If the applied load is to the right of the maximum throughput in Figure 2, then small increases in the mean load lead to a decrease in channel efficiency which in turn increases the backlog in terms of retries and further increases the load. Working to the right of the maximum therefore exposes the system to these instabilities which manifest themselves in variances in the communication bandwidth—a metric we try to minimise [4, 1]. In contrast, when working to the left of the maximum, small increases in the applied load are accompanied by increases in the channel efficiency which helps cope with the increased load and therefore instabilities are unlikely. As an aside, the Ethernet exponential backoff handles the instabilities towards the right by rescheduling failed transmissions further and further into the future, which decreases the applied load.

In *BSPlib*, the mechanism of pacing the transport layer is achieved by using a form of statistical time-division multiplexing that works as follows. The frame size and the number of processors involved in the communication are known. As the processors' clocks are not necessarily synchronised, it is not possible to allow

the processors access in accordance with some permutation, a technique applied successfully in more tightly-coupled architectures [6]. Thus the processors choose a *slot*, q, uniformly at random in the interval $[0 \ldots Q-1]$ (where Q is the number of processors communicating at the end of a particular superstep), and schedule their transmission for this slot. The choice of a random slot is important if the clocks are not synchronised as it ensures that the processors do not repeatedly choose a bad communication schedule. Each processor waits for time $q\varepsilon$ after the start of the *cycle*, where ε is a slot time, before passing another packet to the transport layer. The length of the slot, ε, is chosen based on the maximum time that the slot can occupy the physical medium, and takes into account collisions that might occur when good throughput is being achieved. The mechanism is designed to allow the medium to operate at the steady state that achieves a high throughput. Since the burstiness of communication has been smoothed by this slotting protocol, the erratic behaviour of the low-level protocol is avoided, and a high utilisation of the medium is ensured.

An alternative protocol, not considered here, would be to implement a deterministic token bus protocol in which each station can only send data whilst holding a "token". This scheme was not considered viable as it is inefficient for small amounts of traffic due to the need for the explicit token pass when the token holding processor has no message for the "next to-go" processor. In the worst case this would double the communication time. Also, token mechanisms protect shared resources such as a single Ethernet bus, however a network may be partioned into several independent segments, or processors may be connected via a switch. In this case, the token bus protocol would ensure only a single processor has access to the medium at any time, therefore wasting bandwidth. In contrast, the parameters used in the slotting mechanism can be trivially adjusted to take advantage of a switch based medium. For example, for a full-duplex cross-bar switch, Q can be assumed to be 1 ($Q = 2$ for half-duplex), and ε encapsulates the rate at which the switch and protocol stacks of sender and receiver can absorb messages in a steady state. If the back-plane capacity of the switch is less than the capacity of the sum of the links, then Q and ε can be adjusted accordingly. Therefore, the randomised slotting mechanism is superior to a deterministic token bus scheme, as Q and ε can be used to model a large variety of LAN interconnects.

3 Determining the Value of ε

In any steady state, $T = S$ because, if this were not the case, then either unbounded capacity for the protocol stacks would be required, or the stacks would dispense packets faster than they arrive, and hence contradict the steady-state assumption. Since ε is the slot time, packets are delivered with a mean rate of $1/\varepsilon$ packets per μs. Normalising this with respect to the frame size E gives a value for $T = E/\varepsilon$ packets per unit frame-time. We therefore choose an S value from the curve and infer a value of the slot size $\varepsilon = E/S$ as $S = T$ in a steady state. Choosing a value of $S = 80\%$ and $E = 1200\mu s$ gives a slot size of $1500\mu s$.

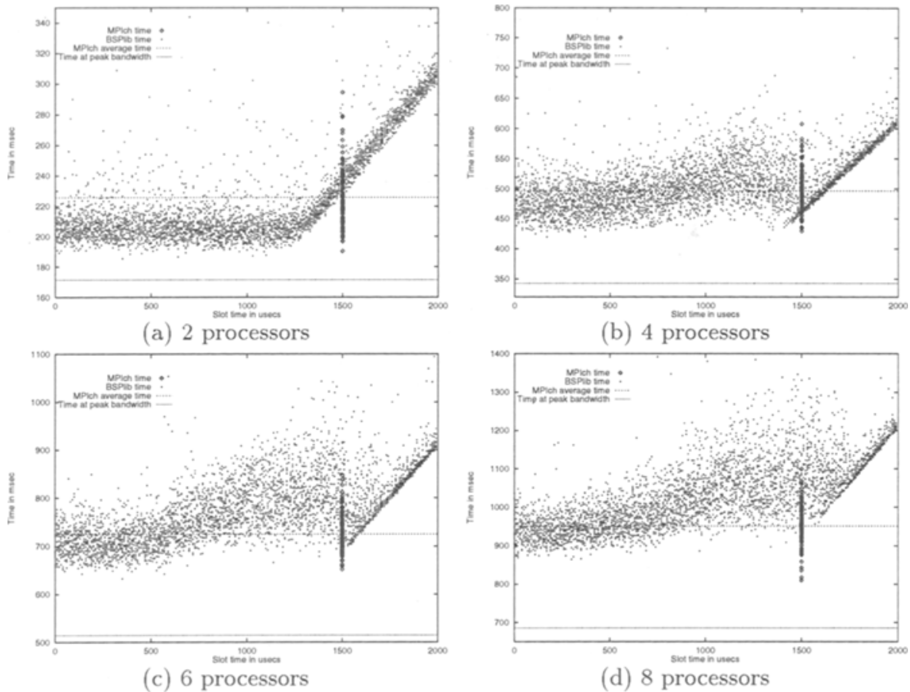
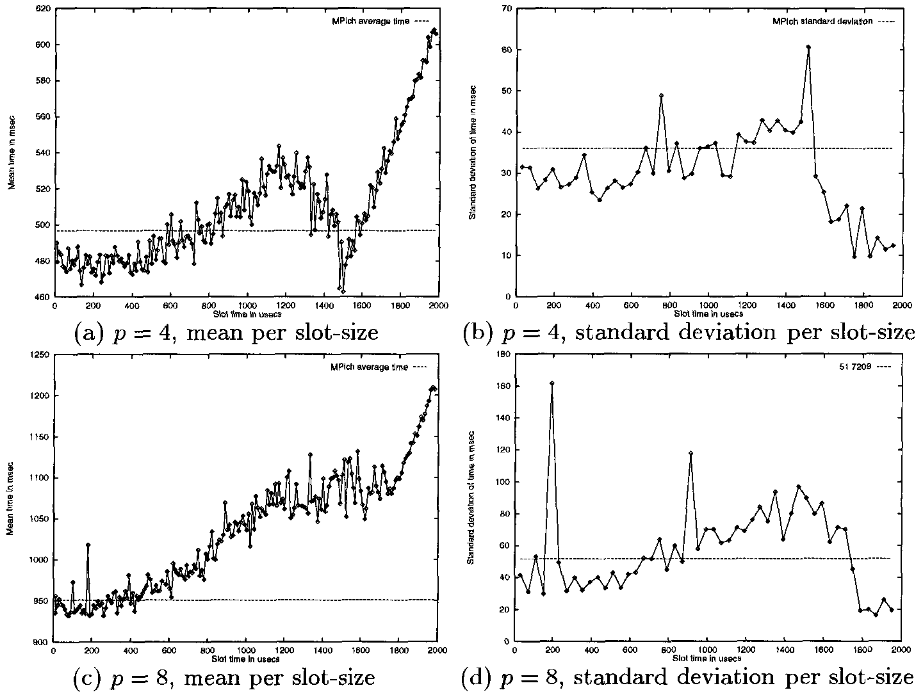


Fig. 4. Delivery time as a function of slot time for a cyclic shift of 25,000 words per processor, $p = 2, 4, 6, 8$, data for `mpich` shown at 1500 although slots are not used

In practice, while the maximum possible value for τ is known, the end-to-end propagation delay of the particular network segment is not, and this influences the slot size via the contention interval modelled in Figure 2. The analytic model assumes a Poisson arrival process, whereas for a finite number of stations, the arrival process is defined by independent Bernoulli trials (the limiting case of this process, as the number of processors increases, is Poisson, and approximates the finite case after the number of processors reaches about 20 [3]). More complicated topologies could also be considered where more than one segment is used.

The slot size ε can be determined empirically by running trials in which the slot size is varied and its effect on throughput measured. The experiments involved a 10Mbps Ethernet networked collection of workstations. Each workstation is a 266MHz Pentium Pro processor with 64MB of memory running Solaris 2. The experiments were carried out using the TCP/IP implementation of *BSPlib*. The machines and network were dedicated to the experiment, although the Ethernet segment was occasionally used for other traffic as it was a subset of a teaching facility. Figure 4(a) to Figure 4(d) plot the time it takes to realise a cyclic-shift communication pattern (each processor **bsp_hpputs** a 25,000 word message into the memory of the processor to its right) for various slot sizes ($\varepsilon \in [0, 2000]$) and for 2, 4, 6 and 8 processors. The figures show the delivery time as a function



(a) $p = 4$, mean per slot-size
(b) $p = 4$, standard deviation per slot-size
(c) $p = 8$, mean per slot-size
(d) $p = 8$, standard deviation per slot-size

Fig. 5. Mean and standard deviation of delivery times of data from Figures 4(a,b)

of slot size, oversampled 10 times. The horizontal line towards the bottom of each graph gives the minimum possible delivery time based on bits transmitted divided by theoretical bandwidth.

Results for an MPI implementation of the same algorithm running on top of the Argonne implementation of MPI (`mpich`) [2] are also shown on these graphs. In this case, the data is presented at a slot size of 1500 μs (even though `mpich` does not slot), so only one (oversampled) measurement is shown. The dotted horizontal line in the centre of these figures is the mean delivery time of the MPI implementation.

The BSP slot time should be chosen to minimise the mean delivery time. Choosing a small slot time gives some good delivery times, but the scatter is large. In practice, a good choice is the smallest slot time for which the scatter is small. For $p = 2$ this is 1200 μs, for $p = 4$ it is 1450 μs, for $p = 6$ it is 1650 μs, and for $p = 8$ it is 1700 μs. Notice that these points do not necessarily provide the minimum delivery times, but they provide the best combination of small delivery times and small variance in these times.

Figure 4(b) shows a particularly interesting case at $\varepsilon = 1500$, as both the mean transfer rate and standard deviation of the *BSPlib* benchmark is much smaller than those of the corresponding `mpich` program. This slot-size can be clearly seen in Figure 5(c) and Figure 5(d) where the scatter caused by the

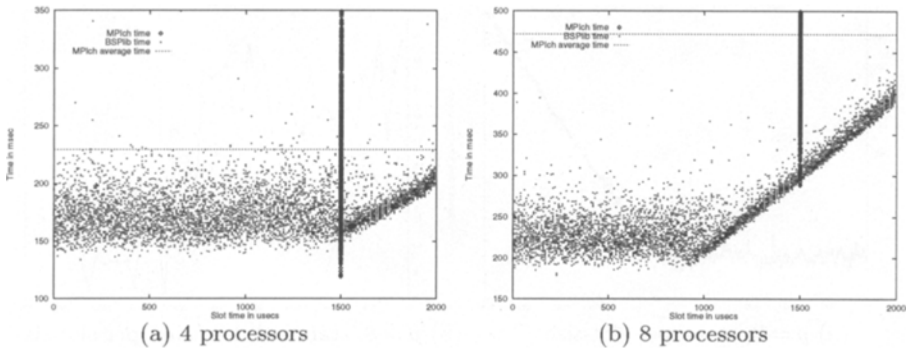


Fig. 6. Delivery time as a function of slot time for a cyclic shift of 8,300 words per processor, $p = 4, 8$, data for `mpich` shown at 1500 although slots are not used

oversampling at each slot size in Figure 4(b) has been removed by only displaying the mean and standard deviation of the oversampling. In contrast, the mean and largest outlier of the `mpich` program in Figure 4(d) is clearly lower than the corresponding *BSPlib* program when a slot size of 1500 is used. For larger configurations, the slot size that gives the best behaviour increases and the mean value of g for *BSPlib* quickly becomes worse than that for `mpich`.

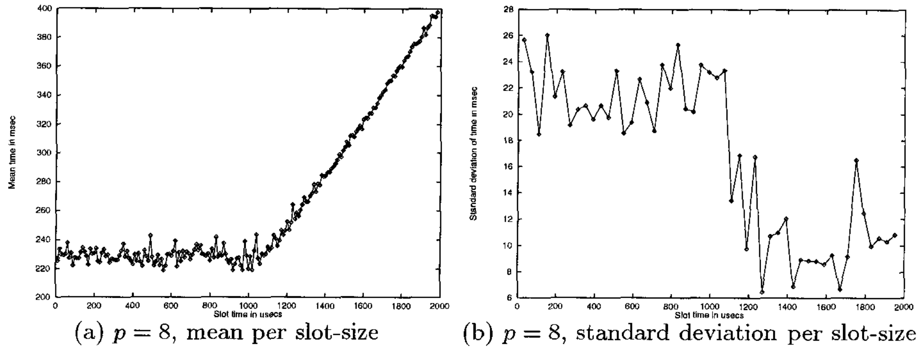
An increase in the best choice of slot size from Figure 4(a) to Figure 4(d) should be expected as the probability $P(n)$ of n processors choosing a particular slot is binomially distributed. Thus as p increases, so does the expectation $E\{X \geq 2\}$ of the amount of contention for the slot, where

$$P(n) = \binom{p}{n}(1/p)^n(1-1/p)^{p-n} \quad \text{and} \quad E\{X \geq 2\} = \sum_{i=2}^{p} iP(i)$$

Figure 3 shows that for $p \approx 20$ and greater, the dependence on p is minimal, and therefore the increase in slot size reaches a fixed point. Below twenty processors the dependence varies by at most 26%. The limit as $p \to \infty$ gives $E\{X \geq 2\} \to 1 - 1/e \approx 0.63$, as shown in the figure. The same is true of the probability of contention, but the range is very small, from 0.25 at $p = 2$, and as $p \to \infty$, $P\{x \geq 2\} \to 1 - 2/e \approx 0.26$.

In the `mpich` implementation [2] of MPI, large communications are presented to the socket layer in a single unit. However, in the *BSPlib* implementation all communications are split into packets containing at most 1418 bytes, so that we can pace the submission of packets using slotting. For this benchmark, each *BSPlib* process sends 71 small packets in contrast to `mpich`'s single large message. Therefore, when p is small we would expect *BSPlib* to perform worse than `mpich` due to the extra passes through the protocol stack, and for larger values of p we would expect that the benefits of slotting out-weigh the extra passes through the protocol stack. Figures 4(a) to 4(d) show an opposite trend.

As can be seen from the Figures 4(b)–(d), as p increases, there is a noticeable "hump" in the data as the slot size increases. This phenomenon is not explained



(a) $p = 8$, mean per slot-size (b) $p = 8$, standard deviation per slot-size

Fig. 7. Mean and standard deviation of delivery times of data from Figure 6(b)

by the discussion above. The problem arises because we are modelling the communication as though it were directly accessing the Ethernet, without taking into account the TCP/IP stack. What we are observing is the TCP acknowledgement packets, which interfere with data traffic as they are not controlled by our slotting mechanism. The effect of this is to increase the optimum slot size to a value that ensures that there is enough extra bandwidth on the medium such that the extra acknowledgement packets do not negatively impact the transmission of data packets.

Implementations of TCP use a delayed acknowledgement scheme where multiple packets can be acknowledged by a single acknowledgement transmission. To minimise delays, a $200ms$ timeout timer is set when TCP receives data [11]. If during this $200ms$ period data is sent in the reverse direction then the pending acknowledgement is piggy-backed onto this data packet, acknowledging all data received since the timer was set. If the timer expires, the data received up to that point is acknowledged in a packet without a payload (a 512 bit packet).

In the benchmark program that determines the optimal slot size, a cyclic shift communication pattern is used. When $p > 2$ there is no reverse traffic during the data exchange upon which to piggy-back acknowledgements. If the entire communication takes less than $200ms$ then only p acknowledgement packets will be generated for each superstep; as the total time exceeds $200ms$, considerably more acknowledgement packets are generated. In Figure 4(a) the communication takes approximately $200ms$ and a minimal number of acknowledgements are generated as can be seen by the lack of a hump. In Figures 4(b)–(d), the size of the humps increases in line with the increased number of acknowledgements. The `mpich` program does not suffer as severely from this artifact as *BSPlib*. When slotting is not used (for example in `mpich`) there is potential for a rapid injection of packets onto the network by a single processor for a single destination, which means that it is likely that more packets arrive at their destination before the delayed acknowledgement timer expires. This reduces the number of acknowledgement packets. When slotting is used, packets are paced onto the network with a mean inter-packet time between the same source-destination pair of $p\varepsilon$. This drasti-

cally decreases the possibility of accumulated delayed acknowledgements. For example, in Figure 4(c), as the total time for communication is approximately 800ms, and as the slot size steadily increases, the number of acknowledgements increases. This in turn steadily increases the standard deviation and mean of the communication time. From the figure it can be seen that this suddenly drops off when the slot size becomes large as the probability of collision decreases due to the under-utilisation of the network.

The global nature of BSP communication means that data acknowledgement and error recovery can be provided at the superstep level as opposed to the packet by packet basis of TCP/IP. By moving to UDP/IP, we can implement acknowledgements and error recovery within the framework of slotting. This lower-level communication scheme is under development, although the hypothesis that it is the acknowledgements limiting the scalability of slotting can be tested by performing a benchmark on a dataset size that requires a total communication time that is less than $200ms$. Figures 6(a)–(d) shows the slotting benchmark for an 8333-relation where there are no obvious humps. In all configurations the mean and standard deviations of the *BSPlib* results are considerably smaller than `mpich`. Also, as can be seen from Figure 7 the optimal slot size at $p = 8$ is approximately $1200\mu s$.

4 Conclusions

We have addressed the ability of the BSP runtime system to improve the performance of shared-media systems using TCP/IP. Using BSP's global perspective on communication allows each processor to pace its transmission to maximise throughput of the system as a whole. We show a significant improvement over MPI on the same problem.

The approach provides high throughput, but also stable throughput because the standard deviation of delivery times is small. This maintains the accuracy of the cost model, and ensures the scalability of systems.

Acknowledgements

The work of Jonathan Hill was supported in part by the EPSRC Portable Software Tools for Parallel Architectures Initiative, as Research Grant GR/K40765 "A BSP Programming Environment", October 1995-September 1998. David Skillicorn is supported in part by the Natural Science and Engineering Research Council of Canada.

References

1. S. R. Donaldson, J. M. D. Hill, and D. B. Skillicorn. Communication performance optimisation requires minimising variance. In *High Perfomance Computing and Networking (HPCN'98)*, Amsterdam, April 1998.
2. W. Gropp and E. Lusk. A high-performance MPI implementation on a shared-memory vector supercomputer. *Parallel Computing*, 22(11):1513–1526, Jan. 1997.

3. J. L. Hammond and P. J. P. O'Reilly. *Performance Analysis of Local Computer Networks.* Addison Wesley, 1987.
4. J. M. D. Hill, S. Donaldson, and D. B. Skillicorn. Stability of communication performance in practice: from the Cray T3E to networks of workstations. Technical Report PRG-TR-33-97, Oxford University Computing Laboratory, October 1997.
5. J. M. D. Hill, B. McColl, D. C. Stefanescu, M. W. Goudreau, K. Lang, S. B. Rao, T. Suel, T. Tsantilas, and R. Bisseling. BSPlib: The BSP Programming Library. *Parallel Computing,* to appear 1998. see `www.bsp-worldwide.org` for more details.
6. J. M. D. Hill and D. B. Skillicorn. Lessons learned from implementing BSP. *Journal of Future Generation Computer Systems,* 13(4–5):327–335, April 1998.
7. P. J. B. King. *Computer and Communication Systems Performance Modelling.* International series in Computer Science. Prentice Hall, 1990.
8. D. B. Skillicorn, J. M. D. Hill, and W. F. McColl. Questions and answers about BSP. *Scientific Programming,* 6(3):249–274, Fall 1997.
9. S. Tasaka. *Performance Analysis of Multiple Access Protocols.* Computer Systems Series. MIT Press, 1986.
10. L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM,* 33(8):103–111, August 1990.
11. G. R. Wright and W. R. Stephens. *TCP/IP Illustrated, Volume 2.* Addison-Wesley, 1995.