Cost Efficient Adaptive Protocol with Buffering for Advanced Mobile Database Applications

S.J. Lai, A. Zaslavsky, G.P. Martin, L.H. Yeo
Department of Computer Technology, Monash University, Melbourne
Email: {SJLai, AZaslavs, GMartin, LHYeo}@broncho.ct.monash.edu.au

Abstract

The most recent advancements in computer hardware and communications make the mobile computing paradigm tangible and feasible. One of the major factors affecting mobile computing is communication protocols efficiency. This paper proposes and discusses an Adaptive Queuing Protocol which targets advanced mobile database applications. The protocol has two main objectives. Firstly, to compensate for the relatively slow speed of some existing mobile communication links. Secondly, to reduce the cost of communications by reducing link usage. In achieving these aims, our goal has been to reduce the total data volume that the link must carry, and at the same time ensure adequate response time for all classes of transactions. Results of computer simulation are presented and discussed.

1. Introduction

Recent advances in miniaturisation and cellular technology make the computing paradigm ubiquitous and are extending the scope of database applications. While distributed databases have been studied and researched for two decades, a new aspect of that same area is now being developed. Distributed databases that incorporate mobile components have opened new avenues of research into advanced applications. Although the technology has been available for 3 years, it has now become affordable, and this has been reflected in the number of new and advanced applications that it is being used for. For instance, in 1993, in the city of Melbourne, Victoria, Australia, taxi companies introduced the concept of a "Computer Cab", a taxi which has an on-board computer that is linked with the database at central headquarters. Police in many countries is equipped now with portable car computers to access information on vehicles and criminals. Eventually, it will be possible for one's pocket computer to communicate automatically with other databases which reside on stationary and/or mobile computers 24 hours a day.

Proceedings of the Fourth International Conference on Database Sytems for Advanced Applications (DASFAA'95)
Ed. Tok Wang Ling and Yoshifumi Masunaga
Singapore, April 10-13, 1995

Operating costs of mobile technology, while shrinking, are still prohibitive for many applications. The capital investment required to obtain a mobile node has dropped dramatically, and may even has reached equilibrium in the market place. An avenue of even greater savings however still remains largely unexplored, and that is the cost of operating the link to/from the mobile node. While there are many flexible pricing schemes, which can reduce the cost of link operation (based on the time of day, number of calls made, etc), we intend to address the fundamental underlying problem, namely, to reduce the amount of data to be transferred.

This paper proposes and discusses an Adaptive Queuing Protocol which targets advanced mobile database applications (a term "mobile databases" represents a class of multidatabase systems where constituent databases may reside on mobile computers and/or stationary computers). This protocol has two main objectives. Firstly, to compensate for the relatively slow speed of some existing mobile communication links. Secondly, to reduce the cost, by reducing link usage. In achieving these aims, our goal has been to reduce the total data volume that the link must carry, and at the same time ensure adequate response time for all classes of transactions.

By the word "link" we mean any method of communication that a mobile platform might use, whether by satellite or mobile phone. We are most concerned with those that have some cost associated with use of the link, rather than those that have no cost that is directly proportional to the amount of data sent (eg., radio modems). For those mobile links that do charge in proportion to the data volume, charges can apply not only on a straight volume basis, but also on a connect time basis.

The remainder of the paper is organised as follows. Section 2 presents a survey of related work and summarises those relatively few references that exist in this highly commercialised and rapidly emerging area. Section 3 discusses the proposed protocol and outlines some associated problems. Section 4 considers a method of obtaining a balanced mix of variables involved in the adaptation process. Section 5 addresses some implementation issues and compares different approaches that have been studied during the implementation phase. Finally, conclusions and a future directions are briefly discussed.

2. Related Work

There have been a number of studies relating to protocol buffering issues [4, 5, 7, 10, 11, 12]. Most, however, tend to have involved relatively high speed local area networks and tend to emphasise performance rather than link usage cost. We denote this class of protocols as Send-on-Demand (SOD) protocols, because they transmit whenever the network is available. Some have investigated the cost of manipulating the buffers used in protocol processing, though once again in an environment where the cost of sending data over the network ranked very low.

Other buffering issues regarding the delayed sending of information can be found in studies related to the implementation of network protocols in operating systems [3, 9]. In these studies some analysis is made on the amount of buffering required for supporting network protocols (TCP/IP, in particular).

Recent research papers [1, 2] demonstrate a growing demand for efficient and effective protocols which can be used in mobile computing. El Abbadi [4] talks about a family of adaptive protocols for maintaining replicated distributed databases. The paper places more emphasis on the integrity of the replicated data rather than the efficiency with which that data is communicated between the distributed databases. Some researchers argue that the major factor which may determine the success or failure of mobile computing is the capacity of a battery and its life-time [8] and under such circumstances the development of efficient energy-saving technologies and robust protocols is especially important. Yeo & Zaslavsky [13] discuss transaction management issues in heterogeneous multidatabase environment with mobile components and propose a queuing mechanism which might be used to increase the efficiency and reliability of mobile applications.

3. Adaptive Queuing Protocol

The adaptive queuing protocol (AQP) has been developed as a result of research which shows that existing network communication protocols for stationary nodes (SN) are sub-optimal when applied to mobile nodes (MN) running database applications. Many existing protocols tend to emphasise the speed of data transfer rather than the cost involved. Viability of MN's, however, is very sensitive to the cost of the link to an SN. AQP attempts to minimise this cost by reducing the number of packets that are transmitted from the MN to the SN across a communications link. AQP does not deal with data flowing in the reverse direction from the SN to the MN.

AQP is built on top of the transport layer if looked at from the viewpoint of the OSI seven layer model. While it assumes that the underlying link is at a transport layer and hence reliable, it does make allowances for the variations in the availability of that link. AQP makes the assumption that a reduction in traffic at higher layers of the path to the network connection, leads to reductions at the lower layers. While some existing distributed database systems attempt to minimise the cost of individual transactions, AQP attempts to minimise the cost of a batch of transactions, viewing them as a whole. Unreliability in the communications channel can cause excessive traffic due to error recovery methods. This situation is not specific to AQP however, and would generally require retransmission of the data regardless of the application type. While AQP could be altered to cater for lower layer error recovery this compromises it's portability and flexibility.

As an example of the saving that can be made, let us assume that our transport layer uses TCP, with IP being used at the network layer. A TCP header is 24 bytes, and a 'typical' IP header is 24 bytes. Assuming that a textual query occupies on average 40 bytes, 6 such queries being sent in rapid succession, each in its own packet, would take 6 x (40 + 24 + 24) = 528 bytes. By collecting all the queries into one packet, however, (6 x 40) + 24 +24 = 288, we make a saving of 240 bytes. While there is a saving in terms of the volume of data sent, we should also consider several factors, namely:

- Some networks charge in terms of the number of packets sent[1], rather than actual data volume. In this case, assuming the aggregated packet in the above example can be sent without fragmentation, there is an ~80% saving to be made. Even if the packet needs to be broken in two, a saving of better than 60% can still be made.
- Battery life has been pointed out as being one of the ultimate determinants of mobile host usage [8].

[1] The MobileData network in Australia is considering an 18 cents per packet charging scheme.

Transmitters of data are one of the most expensive items in terms of battery power. Any reduction in the volume of data to be sent means an extension of battery life and prolonging the operational time of a mobile workstation.

- There are some arguments that on error-prone transmission channels, reducing the time spent in transmitting the data also increases the chances that it will arrive at it's destination error free.

In the AQP model, transactions of varying priorities arrive at the input point, and eventually leave at the output point. When they leave depends on their priority and a number of global system parameters. These parameters can be adjusted while the model is running to adapt it to different situations.

Figure 1 illustrates the AQP model. It consists of a number of input queues, one for each different level of priority that is supported. Although a large number of queues is possible, the task of assigning priorities to transactions in such a situation is difficult. As shown in this diagram priority *n* is the lowest priority, priority 0 is the highest priority.

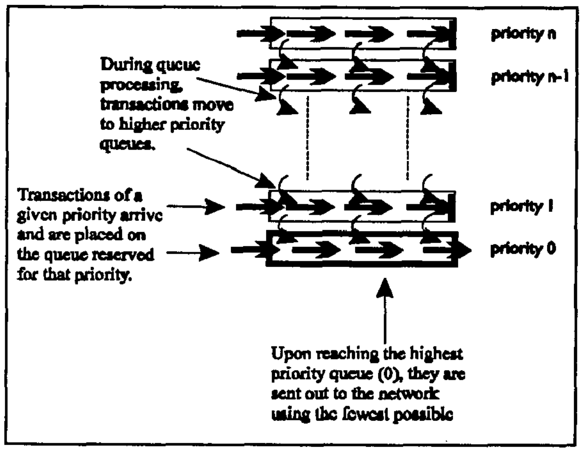
priority n
During queue processing, transactions move to higher priority queues.
priority n-1
Transactions of a given priority arrive and are placed on the queue reserved for that priority.
priority 1
priority 0
Upon reaching the highest priority queue (0), they are sent out to the network using the fewest possible

Fig.1. Architecture of the AQP model

As transactions arrive they are placed on the queue reserved for their particular priority. The queues have no predefined length and are always in a state where they are ready to accept another transaction. A global interval, *freq* (time units), specifies how often the queues are processed. Queues are processed in the following manner:

```
Begin
        $\forall q_i: (q_i \in Q, 1 \le i \le n)$
                $|q_i| + |q_{i-1}| \rightarrow |q_{i-1}|$
                $Empty(q_i)$
End
```

where $|q_i|$ is the number of elements in queue q_i, and Q is the ordered set $\{q_0, q_1, ..., q_n\}$.

This leads to a sort of a "waterfall" effect as the priority of transactions increases. On the way down the waterfall, other transactions are collected and the collection grows until it reaches priority 0. Upon reaching priority 0 transactions wait *keep* * time_units, where *keep* < *freq*. Having *keep* $\geq$ *freq* diminishes the effect of the priorities. When *keep* time_units have expired, all transactions in the priority 0 queue are dispatched by packing them into as few packets as possible. In many situations this may mean one large packet is dispatched, though in others, underlying network limitations will require that a number of smaller packets be sent. Although *keep* is not essential, it avoids priority 0 transactions having to possibly wait for a full *freq* time units before they are dispatched. Their immediate dispatch can be simulated by setting *keep* to 0. A further benefit is apparent in systems that use the concept of virtual calls. By reducing the number of packets and their frequency, a reduction in the number of call setup operations is possible. As call setups are often charged for, this can also reduce operating costs.

Care needs to be taken when choosing the values of *keep* and *freq*. Large values of *freq* mean that transactions at priority *n* will take a relatively long time to be sent. In general, providing *freq* and *keep* to remain constant, a transaction at priority *n* will take at most *(n*freq)+keep* time units to be sent. The value of *keep* is sensitive to the transmission speed and buffering at the mobile end. If the value of *keep* is large and transmission speed slow , the process of sending transactions in priority queue 0 may still be happening when *freq* expires and the transactions from the priority 1 queue 'fall' down into the priority 0 queue. The adaptability of the AQP heavily depends on selecting the optimal values of *keep* and *freq*, and may allow them to be calculated at run-time.

As well as the priority of a transaction, the *AQP* system needs to know whether a response is required from the SN. In most situations this will be true, though in data collection situations where timeliness of data is important, and retransmissions might render the data useless, this might be set to false. The problem of how priorities might be assigned to transactions is the subject of further research, though we assume that priorities roughly equate to response time.

The *AQP* model is relatively cheap to implement. It requires two timers, one for *freq* and one for *keep*. It also requires space for the queues which might be represented in memory or on disk as files. Queues in memory would consist of pointers to incoming transactions. The movement of queue from one priority to another can be implemented as a pointer manipulation. Queues kept on disk provide some degree of protection against power failures. If one file were used for each queue, then moving transactions from one priority to the next could be effected by renaming files (for all except the priority_1 → priority_0 transition). Keeping the queues in files, however, would probably mean that the entire transaction would need to be swapped out to disk.

As well as being cheap to implement, most protocols can implement AQP. Because it is format independent, the internal structure of the protocol does not need to be changed. One area where difficulties might arise, involves the timing constraints of protocols. Depending on how AQP is implemented it might be difficult to determine exactly when a packet will be transmitted. AQP, however, is positioned on top of a transport layer which should reduce the need for timing dependencies in protocols at this layer.

4. Automating the Adaptation Process

Although the application can select the suitable values of *keep* and *freq* manually, a method of feedback to help set these values may be more appropriate. This takes various statistics kept about the queues and uses them to calculate new values of *freq* and *keep*. This is similar to the Quality-Of-Service (QOS) parameters used in the ISO protocols. For example, if a statistic Packet-Per-Minute (PPM) was kept which counted the average PPMs transmitted, this could be used to vary *freq*. On a link which charged for each packet sent, a threshold cost could be set using PPM. If this threshold was exceeded, *freq* could be increased, until PPM came below the threshold. This would provide the system with the best link price/performance ratio. An alternative feedback method might use AVeraGe Delay (AVGD), the average time a packet spends in the priority queues before it is transmitted. When AVGD became too large, *freq* could be reduced.

The feedback method is most useful when the types of query/data being sent across the link varies over time. For constant query/data types, constant optimal values of *freq* and *keep* might be calculated before hand.

With the feedback method, some attention needs to be paid to when the *freq* and *keep* values are changed, and the way the rest of the system interacts with the priority queues. Typically in systems that employ some dynamic adaptation of parameters, there are mechanisms to query the current values of the parameters. If a time-sensitive transaction where to be in the priority queues (such as a real time data measurement sent every 5 seconds), when the values of *freq* and *keep* where changed, the variation in its transmission time (early or late) might render it useless.

5. Implementation of an AQP Simulator

Two simulations of AQP have been developed. Only the second has been used to perform the actual simulations after problems with the first could not be resolved. We describe both below.

5.1 A Multi-Tasking Simulation Technique

This simulation is composed of three processes, contains about 2,000 lines of code, and runs under the Unix operating system. There is a process for each end of a mobile link, and to act as a virtual clock (VC); one of these is responsible for the mobile end of the link (MP - Mobile Process), the other process modelling the stationary end of the link (SP). The processes communicate via the socket mechanism provided by Unix. The architecture of the simulator is shown in Figure 2.

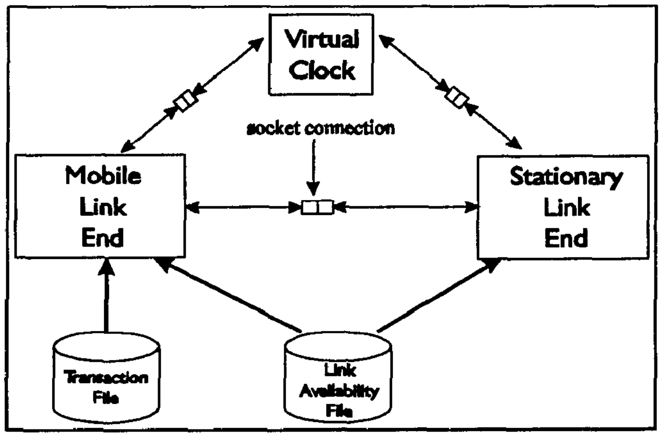

Fig.2. Multi-tasking simulator

The main barrier to the successful functioning of this simulation is the way in which each process receives its time slice from the operating system. Ideally, the SP and MN processes would receive equal amounts of time, and the VC process would be run first (or last) in each iteration. If the time allocated to SP and MN differs, then one appears to race ahead of the other in virtual time, thus skewing the results. If the order of the processes is (SP, VC, MN) rather than (VC, SP, MN), then the SP and MN processes start at different virtual times which makes reconciling the amount of time they consume difficult. Because we did not have detailed control over the scheduling policy in the Unix we used, we determined that the results of this simulation could not be used to support any conclusions which we might draw from them. Newer variants of Unix (OSF/1, Solaris, UnixWare) allow tighter control over the scheduling policy used and might be a suitable environment for such a technique in future research.

Ironically, the independence of the processes, as well as proving a problem, was also the main advantage of this method. The independent nature of the processes allowed them to communicate in an asynchronous manner which most closely approximates the real world.

5.2 Non-Preemptive Multitasking

A non-preemptive multi-tasking simulation is depicted in Figure 3 and is discussed below.

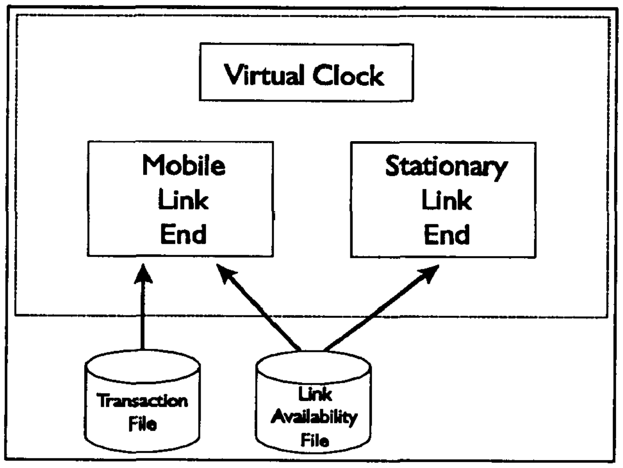

Fig.3. An architecture of a non-preemptive multi-tasking simulator

This simulation uses only one process which causes problems when trying to simulate the independent nature of the different ends of the link. The process runs the Virtual Clock (VC) and contains the simulation code for the Mobile Node (MN) and the Stationary Node (SN). A simple loop, where each iteration simulates a pre-determined number of time_units, might be represented as:

```
Begin Loop
    Update VC
    Run MN code
    Run SN code
End Loop.
```

This however will not work. Although it simulates any transmissions in the MN→SN link resulting from a run of the *Mobile Link End* code, it does not cater for SN→ MN transmissions produced during the same time period. Instead the following loop must be used:

```
Begin Loop
        Update VC
        Run MN code, transmit to SN
        Run SN code, transmit to MN
        Run MN code, receive SN transmissions
        Run SN code, receive MN transmissions
End Loop
```

This allows each end of the link to simulate both receive and send for the pre-determined time period. A few simple rules must be followed by each of the MN and SN code fragments for the simulation to be valid, namely:

- Only those transmissions which fall inside the predetermined time period must occur.
- All transmissions from the other end of the link must be accepted. This assumes that the propagation delay for the link is 0.
- Transmissions which would occur as a result of receiving a transmission from the other end of the link, must be delayed until the next iteration of the loop. Another way of saying this is that the time period to be simulated for each loop iteration must be sufficiently small enough so that this does not happen.

Both simulations are driven by ASCII files, each line of which constitutes a record. The first field of each record contains a timestamp, indicating the starting point of that event in the file. The files are sorted on ascending order based on this timestamp. At this stage there are two main input files, one containing generated transactions that will use the link, and the other listing those times when the link is available. This simulates situations where the link becomes unavailable due to hardware failure (rare), or poor quality of the communications channel (more common). Because the communications channel is assumed to provide a transport layer of service, these breaks in communication are not used to test error recovery but instead to measure the resultant effect on queue length. Certain assumptions have been made to simplify the implementation of the current version of AQP. These assumptions are discussed below.

The other file contains transactions that will use the link for some purposes. As well as the timestamp at the beginning of each record, there are four other white space delimited fields:

- *priority* - the priority of this transaction;

- *cargo* - the size of the data begin carried across the link. This ignores the effects of mechanisms such as compression which can be applied to both this and other protocols with similar results;
- *confirmation* - not all transactions require confirmation. An example of such a situation is presented in one of the case studies below;
- *delay* - associated with the processing of each transaction is an expected delay. Although this is only an estimate, the complexity of the possible networks and processing at the stationary point make accurate calculation (and simulation) of the delay infeasible.

To provide a balanced outcome, a number of different cases have been simulated. Programs customised to a particular case are used to generate the input files for the simulation. These programs are able to generate a specified number of transactions, the attributes of which can be specified to follow a particular statistical distribution (eg. binomial, normal, etc). The first four of the cases that have been run through the simulator are listed below:

- *Constant High Priority Transactions* - This case study favours the traditional SOD method of transmission. Transactions of the highest priority are continually arriving, and being immediately dispatched.
- *Uniform Priority Distribution (UPD)* - In this case, all priorities have an equal chance of occurring. This tends to favour *AQP*, though is not as extreme in its bias as the Constant High Priority Transactions case. Samples of 1000, 10,000 and 1,000,000 were simulated. Results were generally independent of sample size.
- *Travelling Salesperson (TS)* - This case attempts to simulate a travelling salesperson that places orders while on site with clients (stationary order entry). Queries regarding those orders and other stock items are also made while both stationary and mobile. Because links are more reliable when both ends are stationary, the order entry situation suffers less from link unavailability. This salesperson stops and visits 6 clients. At each visit the salesperson performs 10->20 query transactions and finishes with a single write or update transaction.
- *Weather Data Collection Point (WDCP)* - In this case study both ends of the link are stationary and so link availability is quite high. Such a situation might occur when it is prohibitively expensive or impossible to build a land line to the WDCP. The WDCP sends out transactions containing weather information. The frequency of the transactions is non-linear and they do not require a reply. 279 transactions were simulated, representing a fine day, that turns cloudy in the evening.

The length of the simulations varies with the case study. Generally speaking, higher transactions rates require short simulation runs, though a duration thought to be representative of the case study is the over-riding concern.

As well as transmission cost over the link for each simulation a number of other figures are kept. Statistics on queue length (min, avg., max), the number of optimal packets sent vs. the number of sub-optimal packets, time from when a transaction is received to when it is actually transmitted (min, avg., max), as well as others, are also kept.

Generally speaking, the simulation shows a saving in the number of packets sent for *AQP* in all simulations so far. This saving ranges from a few percent for cases biased towards the traditional SOD protocol, to approximately *50%* for cases biased towards *AQP*. As well as absolute transmission cost however, we are also looking at implementation costs, such as the memory consumed by the queues, and the overhead involved in processing the queues.

6. Result Analysis

We have simulated a number of other cases with the protocol simulator though the four mentioned in the previous section are a representative cross section. We do not discuss results for the *Constant High Priority Transactions* case, as this is biased against the AQP and shows no significant savings. The high cost of operating in such a situation over a mobile link would tend to preclude mobile applications that generated a constant stream of high priority transactions.

The results discussed here are for the: UPD, WDCP , and TS. For each of the simulations, two graphs are shown. Total number of packets sent vs. *freq*, and average delay each transaction incurs before it is transmitted vs. *freq*. It should be noted that each of these simulations was chosen to emphasise a particular aspect of the communication behaviour. UPD demonstrates the case where there is a constant stream of transactions each with a randomly chosen (though equally probable) priority. The WDCP illustrates a real time case, and simulates a day of clear weather, followed by some change in the evening. The TS represents a graph which peaks at various times during the day, the peaks representing those times when the salesman is visiting with clients and querying the remote database.

The graphs, depicted in Figure 4, show the results of a UPD simulation of 2000 transactions. The first represents the average number of packets sent vs. different values of *freq*. This graph shows that as *freq* reduces, and hence the effect of the buffering becomes less, the number of packets sent increases, as expected. The most striking aspect of the first graph is that even though we have initiated 2000 transactions, only ~1100 packets are sent. The second graph shows the average packet delay, also for different values of *freq*. What must be remembered when interpreting this graph is that this value represents the average of all packets in an equally probable distribution. In a distribution skewed towards higher priority packets, the send delay would be reduced.

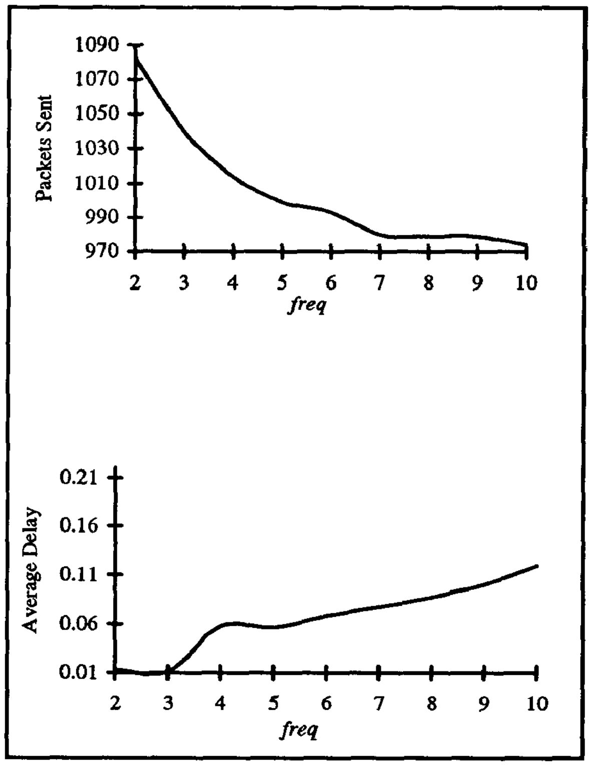

Fig.4. UPD simulation results: Uniform Priority Distribution (2000 transactions)

The results for the weather simulation are depicted in Figure 5, and are not as startling as those for the previous simulation. A total of 279 transactions were conducted during the simulated time. Although a *freq* value of 10 shows a significant saving in terms of the number of packets, it must be remembered that this saving occurred over a whole day at a relatively low transaction rate. Send delay changes almost exactly in geometric proportion with *freq*.

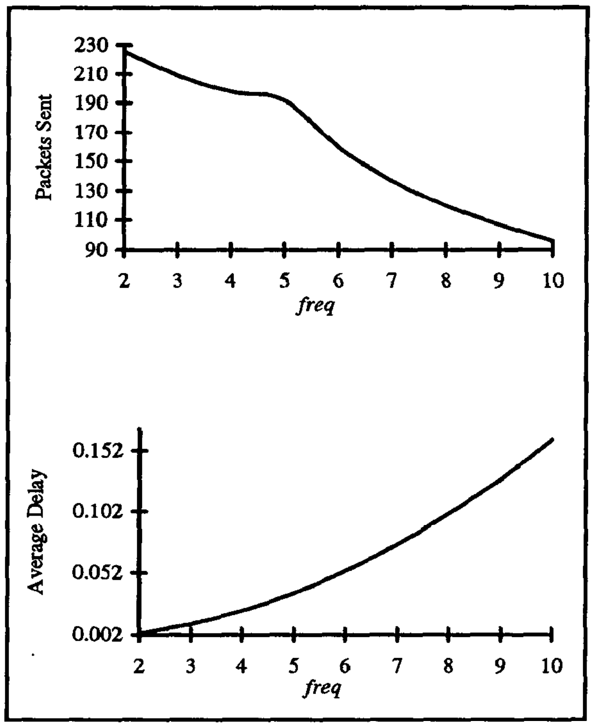

Fig.5. "Weather monitoring" appplication simulation results: Weather Data Collection Point

Travelling salesmen could make a potential saving using AQP. Although the saving at first does not appear to be significant, it must be remembered that the main traffic in this simulation occurs in bursts (during a customer visit) rather than being a constant stream. A snapshot of these high volume bursts yields savings and behaviour similar to the UPD above. A simulation of 96 transactions produced the graphs illustrated in Figure 6.

Simulations were also conducted that varied the time of *keep*. These showed that the average send delay was not greatly affected by varying the value of *keep*. Nonetheless this is an important variable that caters for real-time applications by ensuring the immediate dispatch of packets in the priority 0 queue.

7. Conclusion

Results from simulations of the *AQP* model have shown that for transmission costs it is never worse that the SOD model and in most cases better. Actual savings on packets sent range from a few percent to approximately 50% in the case studies simulated. The simplicity of the protocol lends itself to implementation on portable platforms which are often less powerful, due to power and cost constraints, than stationary nodes. Although it has been developed primarily for use on mobile nodes it is applicable to many other situations where transmission costs are high relative to the information content and priority of the data being sent.

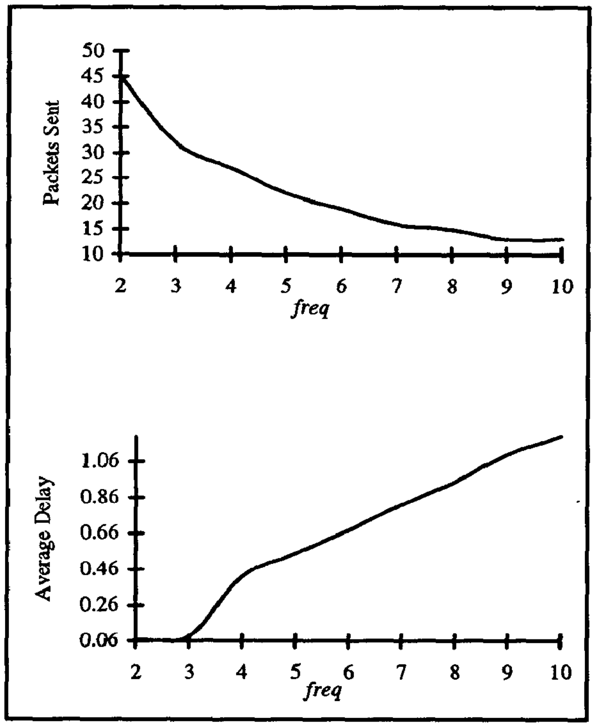

Fig. 6. "Travelling Salesman" application simulation results: Travelling Salesman (6 visits)

This simulation is part of an ongoing study into adaptive supporting mechanisms for protocols. Not only has the number of protocols grown, but also the number of environments in which they are used and tasks to which they are put. Adaptive mechanisms allow protocols to be better tailored to their environment, ultimately benefiting the user. The AQP simulation that has been carried out, is one example of the improvement that can be obtained using adaptive mechanisms. In future work we hope to be able to demonstrate further improvements.

References

[1] Badrinath, B.R., Acharya, A., Imielinski, T., "Structuring Distributed Algorithms for Mobile Hosts", Proceedings of IEEE/CS 14th International Conference on Distributed Computing Systems, Poland, June, 1994, p. 21-28.

[2] Caceres, R., Iftode, L., "The Effects of Mobility on Reliable Transport Protocols", Proceedings of IEEE/CS 14th International Conference on Distributed Computing Systems, Poland, June, 1994, p. 12-20.

[3] Clark, D.C., Jacobson, V., Romkey, J., Salwen, H., "An Analysis of TCP Processing Overhead", IEEE Communications Magazine, June 1989.

[4] El Abbadi, A., "Adaptive Protocols for Managing Replicated Distributed Databases", Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, 1991, p.36-43.

[5] Feldmeier, D.C., "A Data Labelling Technique for High-Performance Protocol Processing and Its Consequences", Computer Communication Review, SIGCOMM'93, Conference Proceedings, Communications Architectures, Protocols and Applications, September 13-17, 1993, San Francisco, California, USA.

[6] Forman, G.H., Zahorjan, J., "The Challenges of Mobile Computing", Computer, Vol. 27, April, 1994, p.38-47.

[7] Huang, Y.M., Guan, S.U., "A Refined Cut-Through Buffer Management Scheme for Layered Protocol Stacks", IEEE Communications, Vol 32, No. 3, March, 1994.

[8] Imielinski, T., Badrinath, B.R., "Data Management for Mobile Computing". SIGMOD Record. 22:1, 1993, p.34-39.

[9] Leffler, S., McKusick, M.K., Karels, M.J., Quarterman, J.S., "The Design and Implementation of the 4.3BSD UNIX Operating System", 1990, Addison--Wesley.

[10] Poo, G.S., Ang, W., "Cut-through buffer management technique for OSI protocol stack", Vol 14, No 3, Computer Communications, April, 1993.

[11] Woodside, C.M., Montealegre, J.R. "On Packet Buffering and Protocol Performance", In: Protocol Specification, Testing and Verification, V, edited by M. Diaz, Elsevier Science Publishers, 1986, B.V.(North-Holland)

[12] Woodside, C.M., Montealegre, J.R., "The Effect of Buffering Strategies on Protocol Execution Performance", IEEE Transactions on Communications, Vol 37, No 6, June, 1989.

[13] Yeo, L.H. and Zaslavsky, A. "Submission of Transactions from Mobile Computers in a Cooperative Multidatabase Processing Environment" Proceedings of IEEE/CS 14th International Conference on Distributed Computing Systems, Poland, June, 1994, p372-379.