Distributed systems support for adaptive mobile applications

Nigel Davies, Adrian Friday, Gordon S. Blair and Keith Cheverst

Distributed Multimedia Research Group, Department of Computing, Lancaster University, UK

Abstract. Mobile applications must operate in environments which experience rapid and significant fluctuations in the Quality of Service (QoS) offered by their underlying communications infrastructure. These fluctuations may be the result of explicit changes between networks, increased competition for network resources or degradation of service due to environmental factors. In order to continue to operate effectively mobile applications must be capable of *adapting* to these changes. This paper reports on the design and implementation of a number of services to support adaptive applications. In particular, the paper describes in detail a Remote Procedure Call protocol (RPC) called QEX which has been designed to adapt to changes in communications QoS and to provide *feedback* to applications when changes to the QoS occur. QEX has been implemented as part of the ANSAware distributed systems platform and together with a number of other services described in this paper enables ANSAware to support advanced mobile applications.

1. Introduction

The development of high-performance portable computers and wireless network technologies has led to the emergence of mobile computing as a major new area of research. In particular, the deployment of local and wide-area wireless networks gives rise to mobile computers which have the potential for continuous network connectivity. However, the actual level of connectivity available to these mobile computers will depend on a number of considerations including their physical location, network availability, environmental factors and (in the case of public wide-area wireless networks) the amount of money users are prepared to pay. In addition, the level of connectivity may change over time as a result of mobility or changes in the environment. Hence, mobile computers must operate in the face of potentially rapid and significant fluctuations in the level of service provided by their underlying communications infrastructure.

Given the fluctuations described above there is, we believe, an emerging requirement for a new class of distributed application designed specifically to operate in this type of environment. Such applications must avoid assumptions about their underlying support environment which may prevent them from operating effectively across a range of networks. Applications of this type are termed *adaptive applications* [14].

The potential of adaptive applications is considerable. Firstly, they can fully exploit the available level of connectivity at any given time. Secondly, adaptive applications are, by definition, more portable. For example, the same applications could be used on a workstation connected to a high speed network and on a mobile computer in the field. Since adaptive applications differ from conventional distributed applications in their ability to exploit changes in the QoS of the underlying communications infrastructure it is important to provide services which enable these applications to monitor and manage the QoS provided by the network. This paper describes a number of services which have been developed at Lancaster to address this requirement. The most significant of these services is a new RPC protocol called QEX which monitors fluctuations in the QoS of the underlying communications channel and adjusts its behaviour accordingly. Moreover, applications using QEX can access the QoS information it gathers in order to adapt their own behaviour to the level of service currently available.

All of the services described in this paper have been implemented as extensions to the ANSAware distributed systems platform [1] which is described in section 2. Section 3 describes in detail the design and implementation of our new RPC protocol QEX. A performance comparison between the existing ANSAware RPC protocol REX and QEX over a range of networks is presented in section 4 together with an analysis of related work. Section 5 describes a number of other services which we have developed to further augment the ANSAware platform and section 6 contains our concluding remarks.

2. The ANSAware distributed systems platform

The ANSAware platform provides programmers with a location-independent object model where all interacting entities are treated uniformly as encapsulated objects. Objects are accessed through operational interfaces which define named operations together with constraints on their invocation. Objects are made available for access by exporting interfaces to a special object known as the *trader*. An object wishing to interact with this interface must then import the interface from the trader by specifying a set of requirements in terms of an interface type and attribute values. This will be matched

against the available services and a suitable candidate selected. At this stage, an implicit *binding* is created to the object supporting the interface, i.e. a communication path is established to the object. Invocation of operations can then proceed.

To provide a platform conformant with the object model, the ANSAware suite augments a general purpose programming language with two additional languages. The first of these is IDL (Interface Definition Language), which allows interfaces to be precisely defined in terms of operations as required by the computational model. The second language, DPL (Distributed Processing Language) is embedded in a host language, such as C, and allows interactions to be specified between programs which implement the behaviour defined by these interfaces. Specifically, DPL statements allow the programmer to *import* and *export* interfaces, and to *invoke* operations in those interfaces. In addition to the trader service described above a number of system services are supplied including a factory service for creating new objects and a node manager for handling object persistence.

In the engineering infrastructure, the binding necessary for invocations is provided by a remote procedure call protocol known as REX or a group execution protocol known as GEX (Group EXecution Protocol). REX is a remote procedure call package which has been designed for two basic styles of interaction: the first, rapid interaction with small amounts of data, and the second, bulk data transfer. REX uses the bulk transfer mechanism when packets exceed a specified fragmentation threshold.

Fig. 1 shows a typical REX RPC interaction (slightly simplified for clarity: for a full explanation see the ANSAware reference manual [1]). Interaction is initiated by the client sending a *call* message to the server. The call consists of a REX message header and a data portion containing all of the information necessary for the server to complete the invocation. The client then periodically retransmits this call until either a *reply* or an *acknowledgement* is received. A reply message consists of a REX header and a data portion with the results of the invocation and is sent by the server as soon as it has finished processing the request. If the server process receives a retransmitted call before it is ready to send a reply it transmits an acknowledgement (consisting of a REX header only) to the client so that the client knows that its message has got through. Reply messages are acknowledged by the client either implicitly by sending a call to the server initiating a new invocation or by an explicit acknowledgement message in response to a retransmitted reply generated by the server.

When the message size is greater than the fragmentation threshold (which is dependent on the transport service), REX fragments the message and sends groups of fragments periodically to the server. The server then informs the client after a fixed interval which fragments haven't been seen (a negative acknowledgement). This style of interaction is depicted in Fig. 2.

REX is layered on top of a generic transport layer interface known as a *Message Passing Service* (MPS). The current version of ANSAware uses either TCP or UDP as message passing protocols but additional protocols may be included at both the MPS and the execution protocol levels and these may be combined to form a number of different configurations. The platform supports lightweight threads within objects allowing multiple concurrent invocations.

All the above engineering functionality is collected into a single library, and an instance of this library is linked with application code to form a *capsule*. Each capsule may implement one or more computational objects. In the UNIX operating system, a capsule corresponds to a single UNIX process. Computational objects always communicate via invocation at the conceptual level but, as may be expected, invocation between objects in the same capsule is actually implemented by straightforward procedure calls rather than by the execution protocols. ANSAware currently runs on a variety of operating

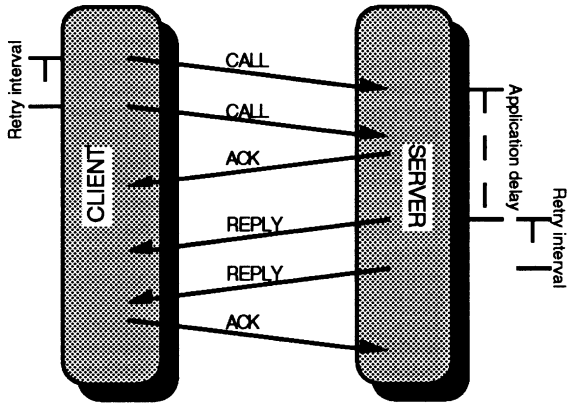

Fig. 1. Simplified REX RPC.

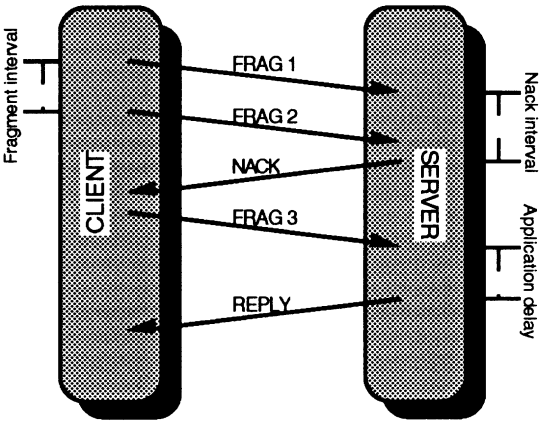

Fig. 2. REX fragmented interaction.

systems platforms including various flavours of UNIX, VMS and MS-DOS/Windows.

3. The QEX protocol

3.1. QEX overview

The current execution protocols in ANSAware are not well suited to operation over mobile networks and offer no support for adaptive applications. More specifically, the REX RPC protocol currently takes no account of the characteristics of the underlying network: parameters such as the number of retry attempts and the interval between these attempts are fixed at installation time. Hence, when a system configured to operate over an Ethernet is run over a low-speed network the unsophisticated congestion control strategy used by REX means that almost no data is actually communicated between user processes. Instead, the network becomes overloaded with REX control messages. If the parameters of the REX protocol are modified and the implementation recompiled to operate over a low speed link it performs poorly over Ethernet.

To enable ANSAware to operate in a mobile environment and to support adaptive applications we have developed a new RPC protocol called QEX which replaces REX but which remains backwardly compatible, allowing applications running either protocol to communicate. QEX analyses the characteristics of the communications medium for each interaction and adjusts itself to make the best use of the link.

The general approach used in QEX is to estimate the underlying characteristics of each channel based on information obtained from round-trip times and sizes of messages. Fig. 1 showed a REX RPC interaction. From this figure we can see that the only pair of messages that could be used to calculate round-trip times is the call/ack or reply/ack interactions since we have no method of determining the application delay (which could change arbitrarily each time, particularly where user interaction is possible) associated with other message pairings. These message pairs, although they occur relatively frequently in most real application scenarios, cannot be relied upon to keep us up to date with the current characteristics of the communications channel. To address this problem, the QEX protocol uses positive acknowledgements whereby objects respond immediately with acknowledgements on receipt of a call or reply message (section 4 discusses the impact of these extra messages on the performance of QEX). These acknowledgements are generated by the protocol libraries prior to the messages being passed up to the application and hence delays attributable to application processing are minimised and a reasonable approximation of round-trip time for a particular message size is obtained (see Fig. 3). The QEX RTT mechanism is analogous to the

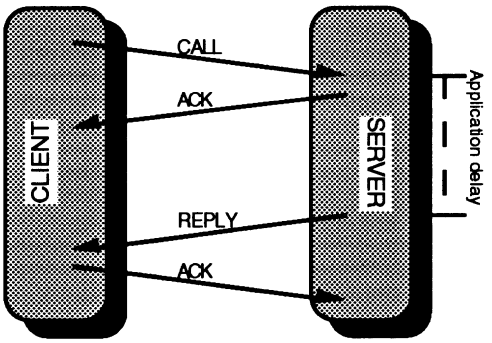

Fig. 3. A simplified QEX RPC.

RTT extensions made to TCP which improve performance in high bandwidth networks [12]. In TCP, a timestamp option field may be added which is echoed back by recipients to enable the sender to calculate the RTT using a single subtraction (additionally, avoiding the aliasing problems identified with earlier TCP window based schemes). In QEX, the timestamp information is kept at the sender and indexed from a single byte modulo counter which is embedded with flag information in QEX packet headers. Like TCP, QEX acknowledgements echo the counter field enabling the sender to index the table and make the RTT calculation. QEX is able to perform the calculation using less transmitted information (the TCP option field is 10 bytes) at the expense of additional processing overhead at the sender.

The round-trip time statistics gathered are smoothed using a moving average calculation and fed back into the QEX protocol to load the retry interval timers. We have been experimenting with a number of different algorithms for approximating throughput and latency based on round-trip times in order to determine the optimal retry interval for a given packet size. Attempts to calculate QoS parameters from packet RTTs are complicated by two factors: firstly, the size of the packets is continually varying (approaches exist for calculating these parameters for continuous media streams, but are based on measuring flows of fixed sized packets or MTUs), and secondly, jitter introduced by error recovery at lower layers (particularly in the transparent services offered by typical wide-area wireless services). Currently, we have implemented two distinct algorithms. The first algorithm simply calculates throughput as a function of packet sizes and round-trip times. This simplistic approach incurs relatively low overheads and in practice works satisfactorily on networks with similar latency characteristics. However, the algorithm's inability to decompose the round-trip times into separate latency and throughput figures makes it inappropriate for use in high-latency networks. The second algorithm attempts to build an internal graph of packet size against round-trip time. This enables us to take account of latency and low-level packet fragmentation which appear as steps in the graph. However, with this

approach it is difficult to determine how changes in round-trip times for a given packet-size should impact on the values recorded for other packet sizes. For example, if the network QoS changes and a 10% increase is observed in the round-trip time for a given packet size, should the estimates for round-trip times for other packet-sizes be adjusted by the same amount? In practice, this algorithm makes better estimations of round-trip times if the network QoS changes infrequently but requires more time to adapt to changes when they occur.

The round-trip time statistics are also used to control the back-off strategy QEX uses. In particular, if the statistics indicate that a low-bandwidth (probably wireless) network is being used QEX adopts a linear backoff strategy rather than an exponential backoff strategy. This decision is based on the assumption that packet loss in low-bandwidth networks is more likely to be due to errors than congestion.

3.2. Dealing with fragmentation

When a message whose size is greater than the fragmentation threshold has to be sent REX fragments the message and sends groups of fragments periodically to the server. The server informs the client which fragments have not arrived after a fixed interval (see Fig. 2). With this approach, there is no scope for working out round-trip times because the negative acknowledgements are generated at the server at fixed intervals and hence we can draw no conclusions from their arrival time.

To overcome this problem QEX explicitly sends negative acknowledgements (indicating which fragments are still to be received) on receipt of fragments which have been tagged by the client. This approach is depicted in Fig. 4. To avoid strict synchronisation during bulk transfer only a proportion of fragments are tagged, the exact number being based on the stability of the channel measurements. The number of fragments sent each period is dependent on the current estimate of the underlying network characteristics.

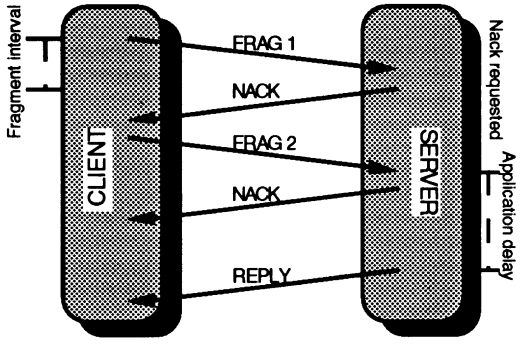

Fig. 4. QEX fragmented interaction.

3.3. QoS information and usage

In addition to maintaining and using QoS information to improve its performance QEX also provides application programmers with a number QoS related functions in order to enable the development of adaptive applications. These functions are available for each binding between objects and are accessed via a binding control interface which is created explicitly by the programmer. Note that the creation of explicit bindings is in contrast to the approach adopted in vanilla ANSAware where all bindings are created implicitly.

The QoS parameters associated with each binding are: the throughout, the maximum *idle times* for the client and server interfaces and a number of characteristics associated with the cost of the channel (e.g. tariff structure). The idle time of an interface is the time that has elapsed since the interface last sent or received a message (the use of idle times is explained later in this section).

The binding controller interface allows programmers to carry out the following operations:

- Obtain the QoS of the binding, i.e. return the current values of the above QoS parameters.
- Set the QoS of the binding. This operation allows programmers to specify the desired QoS of the binding. Clearly, QoS can only be guaranteed in an environment with real-time end-systems and networks which support reservations of bandwidth. In our system QoS parameters set by the application programmer are simply used as hints to help prioritise QEX operations.
- Register for changes in the QoS. Programmers can register for changes in any of the QoS parameters or violations of the idle time thresholds.
- Delete the binding (unbind).

For example, using the binder control interface a programmer can monitor the idle time of interfaces without having to explicitly send application level test messages, i.e. applications can delegate responsibility for guaranteeing QoS assertions to the system. This is of significance since it allows mobile applications to be structured in an event based fashion (cf. polling). Thus, through the use of our bindings, it is possible to assert that the absence of messages on a given interface for a given period of time (the elapsed idle time) is a result of there being no traffic intended for the specified interface rather than a result of communications failure. In addition, QoS driven bindings allow the system to optimise the use of test messages which might otherwise be duplicated if left to individual applications, e.g. if multiple applications wished to test QoS assertions between the same pair of objects.

It should be stressed that in order to reduce the overhead associated with their creation, bindings are estab-

lished independently at the client and server. Indeed, a physical communications channel between client and server need not be in place at the time of binding.

3.4. Implementation status

The QEX protocol has been fully implemented and integrated into version 4.1 of ANSAware. The resulting platform has been demonstrated running in a heterogeneous environment consisting of SUN workstations, desktop PCs and portable PCs. Mobile communication is provided by either analogue cellular phones, operating at 2.4 Kbits/sec, or GSM at 9.6 Kbits/sec. In addition, a network emulator [8] has been used to allow us to simulate dynamically varying network QoS. The QEX protocol has been implemented to run over a UDP like protocol (which we call Serial-UDP) which handles serial and Hayes compatible dialup connections.

In addition to strategies for connection management we are currently experimenting with using the Serial-UDP implementation to schedule messages based on deadlines which can be assigned at the application level via the QEX programming interface. Deadlines provide a useful mechanism for specifying the relative importance of messages and allow interactive traffic to take precedence over background traffic such as cache updates. Furthermore, in the case of dial-up connections deadlines can be used to help determine the optimum times to establish and break these connections. Deadlines have the advantage that they avoid starvation as is often found in priority based systems (as reported in [10] and addressed using a probabilistic approach based on lottery scheduling [17]). However, deadlines have the drawback that background traffic becomes increasingly important over time and heavily delayed background traffic may eventually be transmitted in preference to more recent interactive traffic. We intend to address this issue using a hybrid scheme incorporating both deadlines and priorities.

4. Evaluation

4.1. Performance

This section analyses the performance of the QEX protocol and compares it to REX. The results are based on experiments conducted using a network emulator (as mentioned in section 3.4) to enable us to simulate a wide range of network qualities of service.

Experiment 1: Comparing REX and QEX (no fragmentation)

The first experiment compares the performance of QEX and REX for a variety of network bandwidths. The packet size is kept constant (115 bytes) and below the level of fragmentation. Fig. 5(a) shows the number of

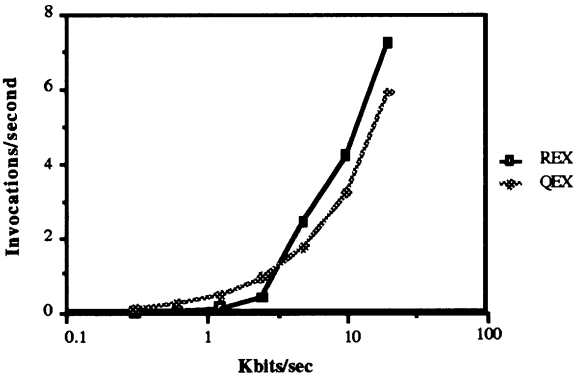

Fig. 5(a). Comparison of throughput against data rates.

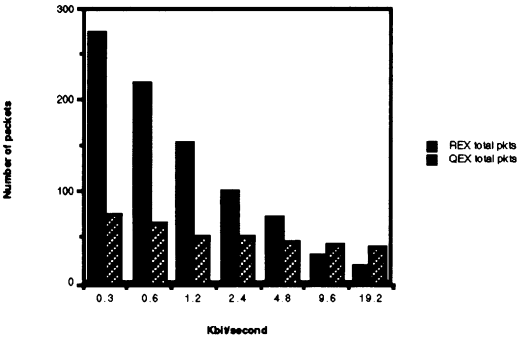

Fig. 5(b). Number of messages sent to complete ten invocations.

invocations per second for REX and QEX for a range of network bandwidths. The graph shows that QEX performs significantly better than REX at low data rates (up to approximately 4.8 Kbits/sec) but that the situation is reversed at higher data rates. The reason for the better performance of QEX at low bandwidths is illustrated in Fig. 5(b) which shows the number of messages each protocol sends to transmit ten invocations for the same range of bandwidths. The large number of messages REX sends at low bandwidths reflects the inappropriate retry interval, which is tuned for higher-speed networks. At higher bandwidths the retry intervals are more appropriate and REX is able to outperform QEX because of the overheads QEX incurs in gathering QoS information. In particular, when message transmission time is small (small packet size or fast network) and application delay is very low, REX receives the reply from the server before the retry time-out and so the retry/ack interaction is unnecessary. If there is a subsequent call to the same server waiting at the client then it can be transmitted immediately on receipt of this reply. This second call informs the server that its reply was received thus removing the need for an explicit acknowledgement message. In contrast, QEX always performs explicit acknowledgements to allow it to assess round-trip times. Therefore, in experiment 1, with repeated invocations and low message transmission times REX uses less messages than QEX.

It should be noted that this experiment highlights the

worst case behaviour of QEX, where a single client repeatedly sends small packets to a single server who replies with minimal delay. Object interactions in real applications tend to be more sporadic and application delays are more pronounced; in such cases REX is unable to use the optimisation described above.

Experiment 2: Comparing REX and QEX (with fragmentation)

The second experiment examines the performance of QEX and REX with an invocation payload that causes the fragmentation mechanism to be used. The amount of data transferred each time was 2Kbytes. Fig. 6(a) plots the number of invocations achieved per minute against varying bandwidths. This figure shows that QEX consistently works better than REX. The reason for this is illustrated in Fig. 6(b) which shows the actual number of fragments sent by REX when attempting to send a two fragment invocation for different data rates. After one invocation at 2.4Kbits/sec, REX has sent and queued over 590 fragments, a second invocation would not succeed because of the resulting congestion. Below this rate, the first invocation times out before a single request/reply can get through (over 6,000 fragments waiting to be sent). The REX fragmentation mechanism does not operate any form of congestion control once invocations become fragmented. In contrast, QEX maintains a consistent rate of two fragments (i.e. the minimum possible) over all the data rates tested once the process of adapting is completed.

Experiment 3: Reaction to fluctuations in QoS

The third experiment shows the speed at which QEX adapts to changes in channel bandwidth. We measure the number of unnecessary retries before the protocol adapts to the new bandwidth for different fractional rate changes (a fractional change of 1/2 implies the bandwidth is dropped to a half of its original value). The figure is calculated by averaging over different ranges (i.e. 9.6–4.8Kbits/sec, 4.8–2.4Kbits/sec, etc.). The largest drop in rate is where QEX adapts from the default rate of REX to a 300 bits/sec channel.

Fig. 7(a) illustrates adaptation for non-fragmented invocations with identical size packets to those used in experiment 1. As the change in rate is instantaneous, the protocol continues transmitting at the current rate until a change in round-trip time is detected. Once the protocol realises the rate has changed it begins adapting to the new rate, so less extra packets are transmitted for the next invocation, and so on. The figure shows the total number of unnecessary reties sent while QEX is adapting. This total figure is composed of retries sent during

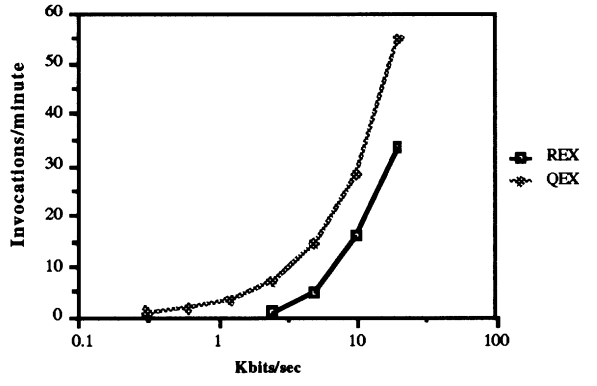

Fig. 6(a). Comparison of throughputs achieved by REX and QEX for different data rates.

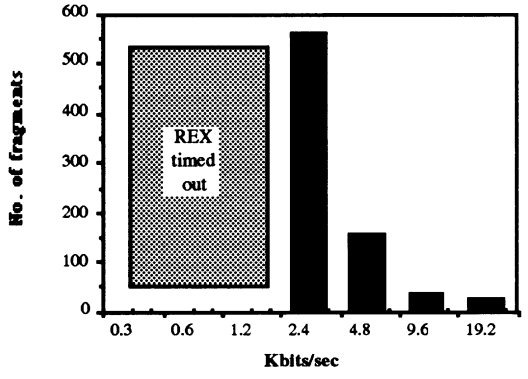

Fig. 6(b). Fragments sent by REX for different data rates.

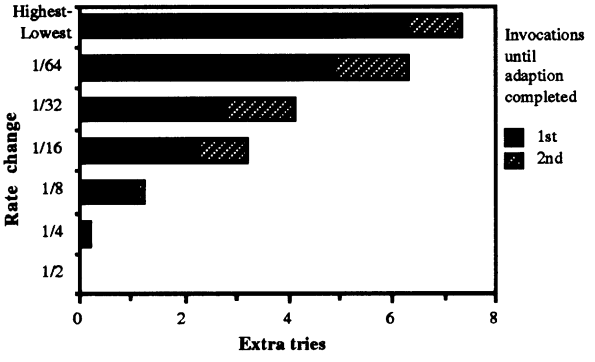

Fig. 7(a). Extra tries per invocation until adaptation completed.

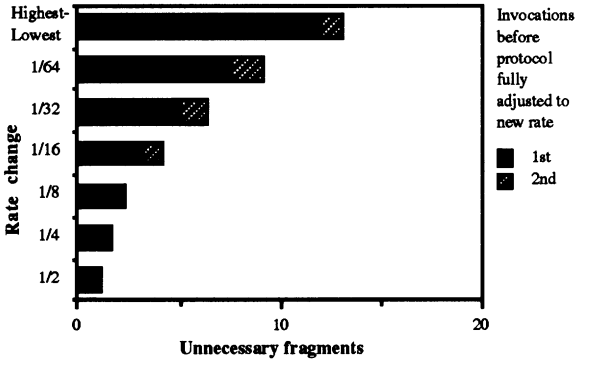

Fig. 7(b). Extra fragments per invocation until adaptation completed.

the first two invocations following the change in rate (for the changes tested, no more than two invocations are needed before the protocol has fully adapted to the new rate). From Fig. 7(a) we can see that where bandwidth halves, no unnecessary retries occur in adapting to the new rate. When bandwidth drops to an eighth of the former rate, on average one unnecessary retry is sent during the first invocation. Over the maximum drop in bandwidth no more than 8 unnecessary retries are sent with approximately six of those occurring in the first invocation.

Fig. 7(b) illustrates adaptation for fragmented invocations (with a payload size of 2Kbytes as used in experiment 2). As for non-fragmented adaptation, the protocol continues transmitting at the current rate until a change in round-trip time is detected. Once the protocol realises the rate has changed, it begins adapting to the new rate, requesting explicit negative acknowledgements after every fragment until it has sufficient round-trip times to transmit no redundant fragments. Once the rate becomes stable the tight fragment/nack coupling is relaxed over time. Although extra fragments are transmitted while the protocol is adapting, these are only redundant if they are unnecessary retransmissions of fragments that the server has already received. If the change in bandwidth occurs during a long invocation (with many fragments) then these fragments would have to be transmitted anyway and can be treated as congestion with the usual backoff strategy. As with non-fragmented invocations, the protocol needs no more than two invocations to complete adaptation in all the changes tested.

4.2. Related work

Related research has been carried out in the areas of adaptive services [7], transport protocols for mobile hosts [3,4,18] and mobile IP [13]. However, the majority of this research is based on the premise that mobility should be hidden with levels of abstraction. Thus, in theory, allowing existing applications and services to run with minimal modification. In contrast, QEX provides facilities for explicitly informing applications of changes in the QoS of the underlying communications channel.

There are a number of RPC protocols which have been adapted to operate over a range of networks. For instance, the efforts of Bachmann et al. [2] in modifying Rx, the RPC package which underpins AFS, to run over low-bandwidth SLIP lines. Rx is a general purpose RPC package which is widely used in the internet and which offers a streaming interface to the service layer, enabling single RPC file transfers. Rx is based on UDP and is thus required to implement its own flow-control, packet fragmentation and reassembly, and authentication schemes. In common with TCP, Rx has been modified to adjust its retransmission timers based on a measure of RTT and variance [11] (actually an approximation based on the mean deviation that is simpler to calculate). Rx is able to operate over a range of networks, unlike QEX however, it provides no managed information to higher layers and therefore offers no support for adaptation to applications. QoS management functionality could be added to Rx, although the new mechanisms would necessitate further work in developing a suitable API as Rx is not part of a distributed systems platform.

There has been a considerable body of work on transport protocols which can operate in a mobile environment. Cáceres and Iftode demonstrated in [5] that vanilla TCP was not suitable for use in an environment where packet loss was due to factors other than congestion (e.g. as a result of high bit-error rates or cell hand-offs). This is because TCP makes the assumption that packet loss is generally as a result of congestion and so backs off exponentially to avoid flooding the network and causing further packet loss. Exponential backoff has been shown to be an excellent strategy in fixed networks. However, in a mobile environment where packet loss is often due to bit-errors and hand-offs this strategy leads to very poor channel utilisation. Cáceres and Iftode addressed this problem by modifying TCP such that it carried out a fast retransmission when signalled by the underlying mobile IP implementation that packet loss was due to a recent cell hand-off. Such signals do not, of course, help when packet loss is due to bit-errors but is consistent with our thesis that issues of mobility should not be hidden. By comparison, QEX does not rely upon signals from the underlying network but instead dynamically adjusts its back-off strategy based on the observed characteristics of the channel.

Other work on transport protocols for mobile hosts includes I-TCP [3], MTCP [18] and M-RPC [4], all of which attempt to provide satisfactory performance by splitting each end-to-end connection into a number of separate connections. Each connection can then be implemented using a protocol tuned for the characteristics of the underlying network. Such approaches suffer from the need for intermediate agents which manage wireless connections. Furthermore, all of the approaches described above attempt to provide the same programming interface as TCP, masking out the effects of mobility from higher levels. This is in direct contrast to the approach adopted in QEX which, as mentioned previously, is to make QoS information easily accessible to higher levels.

The QEX binding interface, through which applications obtain QoS information, is based loosely on the API for bindings described in RM-ODP and associated work on QoS bindings for multimedia traffic [6]. Within the field of mobile computing, the QEX approach can be compared to the work of Schilit et al. [16] who designed an API for context aware applications and the work of Noble et al. [15] on an API for an adaptive file system. Schilit's API is based on the notion of dynamic envi-

ronment variables which change to reflect, for example, a mobile computer's physical location. These changes can be used as triggers for application adaptation in much the same way as QEX call-backs. While QEX could use environment variables to notify applications of QoS changes the use of a separate functional interface allows applications to control QoS as well as monitor changes. The work of Noble et al. has concentrated on designing a QoS based API for the Odyssey file system rather than for a general purpose distributed systems platform.

5. Additional services

5.1. G-QEX

In addition to QEX we have designed, but not implemented, a group execution protocol called G-QEX (Group Quality-of-service EXecution protocol) to further augment the ANSAware platform. G-QEX replaces ANSAware's standard group execution protocol GEX and offers support for adaptive applications requiring group communications. In particular, G-QEX is designed to allow the group transparency paradigm to be selectively maintained, partially broken or completely broken at the application level. In order to retain complete group transparency a client of the group sends a message in the normal way and G-QEX will propagate the message to the group using a default policy. Transparency can be partially or totally discarded by establishing an explicit binding between the client and the group interface with an associated binding control interface (as described in section 3.3). Through the binding control interface clients will be able to choose to partially discard group transparency by specifying the quorum to be used for deciding whether or not an invocation on the group has been successful, or to totally discard group transparency by specifying a *message profile* stipulating the required QoS to be used when propagating their next group invocation. The message profile is a matrix associating group members with a set of QoS (Quality Of Service) parameters. The parameters we currently envisage supporting are:

(i) Temporal constraints which stipulate the time out period within which the group member must acknowledge receipt of the group invocation.

(ii) Ordering requirements which stipulate whether or not the group member must receive group invocations in sequence.

(iii) Reliability requirements which stipulate whether or not the group member must receive the group invocation.

(iv) Cost requirements which stipulate the cost which the client is prepared to pay in order to have the group member receive the group invocation.

An example of the message profile structure is shown in Fig. 8 where the first column represents a group member's id, the second column represents the time constraint, the third column represents the required ordering, the fourth column represents the reliability guarantee and the last column represents the cost in an appropriate unit.

There will be occasions when group membership is increased before the client is able to update the message profile to take account of the new group members. In this situation, the current message profile will be updated automatically by using a set of default QoS parameters. The client will be able to stipulate these default parameters by entering them in the first row of the message profile.

If a client's group invocation fails due to a QoS violation then this information will be reported back to the client via an appropriate error code. The client can then interrogate the binding to establish the cause of the failure. Note that not all QoS violations will result in overall failure of the invocation due to factors such as low quorums. However, clients will be able to register an interest in QoS violations which will enable them to be informed of all QoS violations.

G-QEX will, in the first instance, be engineered using QEX as an underlying transport protocol. This will enable G-QEX to operate in a heterogeneous network environment and will allow us to use QEX's QoS information to provide G-QEX's support for message profiles and QoS.

5.2. Trading services

In addition to QEX and G-QEX we have modified the ANSAware distributed systems platform to include a new trading service. The implementation of the trading function in ANSAware assumes a hierarchical pattern of linking. This hierarchical arrangement while possibly adequate for fixed networks is inappropriate for mobile systems where each portable computer is likely to have its own trader in order that it can continue operation during periods of disconnection. Moreover, given the nature of the communications network a link scheme which requires traversal up and down a hierarchy (possibly requiring multiple dial-up connections to be established) is clearly unsuitable.

We have addressed this issue by introducing a

MemberId	Time	Ordering	Reliability	Cost
0,	2,	ordered,	yes,	20,
1,	2,	ordered,	yes,	20,
2,	2,	unordered,	no,	20,
3,	2,	unordered,	no,	20

Fig. 8. An example message profile.

mechanism to enable peer-to-peer linking of traders. The links thus established can have constraints imposed on them ensuring that they are traversed only when there is a strong likelihood of the remote trader being able to satisfy the request. For example, we can specify that services owned by a particular user are always found on a specified machine and traverse the link to that machine only when looking for that user's services. Furthermore traders can prioritise their links to other traders and hence determine the order in which links are traversed. Link priorities are dynamic and hence can be adjusted on the basis of changes in network QoS allowing traders to optimise their search for services which match a client request.

6. Concluding remarks

This paper has reported on the design and implementation of a number of services which augment the ANSAware distributed systems platform and allow it to support adaptive mobile applications. The key service described was the QEX RPC protocol which is designed to operate in environments in which there are rapid and significant fluctuations in QoS in the underlying communications channel. QEX is able to monitor the QoS of the communications channel and provide this information to applications on request to enable them to adapt their behaviour. The results of a performance comparison between REX and QEX have been presented. This comparison showed that while there are some specific cases where the overhead of QoS monitoring means that REX out-performs QEX in terms of invocation times, over low-speed networks QEX has substantially lower invocation times. While we would not claim to have produced an optimised implementation of QEX we do believe the implementation demonstrates important concepts and we intend to use QEX mechanisms and the associated programming interface as part of a wider QoS architecture which will encompass additional QoS parameters including cost and power.

We have used the ANSAware platform with QEX and our new trading services to build a number of prototype adaptive applications including a structured email system, a database access program and a collaborative tool for annotating geographical information. All of these applications use QEX as their only means of communication and have been demonstrated operating over both wired and wireless networks. In addition, the applications use the QEX API to adapt to changes in the QoS of the underlying communications channel by, for example, varying the type and amount of information returned in response to database queries or by providing collaborating users with feedback so that they can match their mode of working to the level of support available. Further details on these adaptive services can be found in [9].

References

[1] A.P.M. Ltd., ANSAware 4.1 application programming in ANSAware, Document RM.102.02, A.P.M. Cambridge Limited, Poseidon House, Castle Park, Cambridge CB3 0RD, UK (February 1993).

[2] D. Bachmann, P. Honeyman and L.B. Huston, The Rx Hex, *Proc. 1st International Workshop on Services in Distributed and Networked Environments*, Prague, Czech Republic (June 1994) (IEEE Computer Society Press) pp. 66–74.

[3] A. Bakre and B.R. Badrinath. I-TCP: Indirect TCP for mobile hosts, *Proc. 15th International Conference on Distributed Computing Systems*, Vancouver, Canada (May 1995) pp. 136–143.

[4] A. Bakre and B.R. Badrinath, M-RPC: A remote procedure call service for mobile clients, Technical Report WINLAB TR-98, Department of Computer Science, Rutgers University, U.S. (June 1995).

[5] R. Cáceres and L. Iftode, The effects of mobility on reliable transport protocols, *Proc. 14th International Conference on Distributed Computer Systems*, Poznan, Poland (June 1994) pp. 12–20.

[6] G. Coulson, G.S. Blair, J.B. Stefani, F. Horn and L. Hazard, Supporting the real-time requirements of continuous media in open distributed processing, *Comp. Networks and ISDN Syst.* 27(8) (Special Issue on ISO Reference Model for Open Distributed Processing) (1995) 1231–1246.

[7] N. Davies, S. Pink and G.S. Blair, Services to support distributed applications in a mobile environment, *Proc. 1st International Workshop on Services in Distributed and Networked Environments*, Prague, Czech Republic (June 1994) pp. 84–89.

[8] N. Davies, G.S. Blair, K. Cheverst and A. Friday, A network emulator to support the development of adaptive applications, *Proc. 2nd USENIX Symposium on Mobile and Location Independent Computing*, Ann Arbor, US, pp. 47–55.

[9] N. Davies, G.S. Blair, K. Cheverst and A. Friday, Experiences of using RM-ODP to build advanced mobile applications, *Distributed Syst. Eng. J.* 2(3) (1995) 142–151.

[10] L.B. Huston and P. Honeyman, Partially connnected operation, *Proc. 2nd USENIX Symposium on Mobile and Location-Independent Computing*, Ann Arbor, US, pp. 91–97.

[11] V. Jacobson, Congestion avoidance and control, *Proc. ACM SIGCOMM* (August 88) pp. 314–329.

[12] V. Jacobson, R. Braden and D. Borman, TCP extensions for high performance, *RFC 1323*, NWG (May 92).

[13] D. Johnson, Routing in ad hoc networks of mobile hosts, *Proc. Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US (December 1994).

[14] R.H. Katz, Adaption and mobility in wireless information systems, *IEEE Personal Commun.* 1(1) (1994) 6–17.

[15] B.D. Noble, M. Price and M. Satyanarayanan. A programming interface for application-aware adaptation in mobile computing, *Proc. 2nd Usenix Symposium on Mobile and Location Independent Computing*, Ann Arbor, US, pp. 57–66.

[16] B. Schilit, N. Adams and R. Want. Context-aware computing applications, *Proc. Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US (December 1994) pp. 85–90.

[17] C.A. Waldspurger and W.E. Weihl, Lottery scheduling: Flexible proportional-share resource management, *Proc. First Symposium on Operating Systems Design and Implementation*, Monterey, California, US (November 1994) pp. 1–11.

[18] R. Yavatkar and N. Bhagwat, Improving end-to-end performance of TCP over mobile internetworks, *Proc. Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US (December 1994) pp. 146–152.

Nigel Davies graduated from Lancaster University in 1989 and later that year joined the department as a research associate investigating storage and management aspects of multimedia systems. As a result of his work in this area he was awarded a Ph.D. in 1994. After a spell as a visiting researcher at the Swedish Institute of Computer Science (SICS), where he worked on mobile file systems, he returned to Lancaster, first as site-manager for the MOST mobile computing project and subsequently as a lecturer in the Computing Department. His current research interests include mobile computing, distributed systems platforms and systems support for multimedia communications. Nigel is a member of the ACM.
E-mail: nigel@comp.lancs.ac.uk

Adrian Friday graduated from the University of London in 1991. In 1992 he joined the mobile computing group at Lancaster University, working towards his Ph.D. on "Infrastructure support for adaptive mobile applications" under the auspices of the MOST project. In 1995 he became a research assistant investigating distributed systems support for advanced applications within the context of the Mobile Multimedia project funded by the EPSRC/DTI sponsored LINK/PCP programme.
E-mail: adrian@comp.lancs.ac.uk

Gordon Blair is currently a professor in the Computing Department at Lancaster University and a member of the ACM. He completed his Ph.D. in computing at Strathclyde University in 1983. Since then, he was an SERC Research Fellow at Lancaster University before taking up a lecturership in 1986. He has been responsible for a number of research projects at Lancaster in the areas of distributed systems and multimedia support and has published over a hundred papers in his field. His current research interests include distributed multimedia computing, operating system support for continuous media, the impact of mobility on distributed systems and the use of formal methods in distributed system development.
E-mail: gordon@comp.lancs.ac.uk

Keith Cheverst is a research assistant with the Computing Department at Lancaster University working on a project concerned with research into reactive services for mobile environments. This project builds on the research issues identified in the MOST project with which he was formally associated. He is also currently involved in research for his Ph.D. which focuses on the special requirements of groupware applications designed to operate in weakly connected environments. In particular, he is concentrating on establishing ways of increasing the dependability of groupware applications.
E-mail: kc@comp.lancs.ac.uk