Adaptive Forward Error Correction for Real-Time Groupware

Jeff Dyck, Carl Gutwin, and Dwight Makaroff

Department of Computer Science, University of Saskatchewan
110 Science Place, Saskatoon, Saskatchewan, Canada

[jeff.dyck, carl.gutwin, dwight.makaroff] @usask.ca

ABSTRACT

Real-time distributed groupware sends several kinds of messages with varying quality-of-service requirements. However, standard network protocols do not provide the flexibility needed to support these different requirements (either providing too much reliability or too little), leading to poor performance on real-world networks. To address this problem, we investigated the use of an application-level networking technique called adaptive forward error correction (AFEC) for real-time groupware. AFEC can maintain a predefined level of reliability while avoiding the overhead of packet acknowledgement or retransmission. We analysed the requirements of typical real-time groupware systems and developed an AFEC technique to meet these needs. We tested the new technique in an experiment that measured message reliability and latency using TCP, plain UDP, UDP with non-adaptive FEC, and UDP with our AFEC scheme, under several simulated network conditions. Our results show that for awareness messages that can tolerate some loss, FEC approaches keep latency at nearly the plain-UDP level while dramatically improving reliability. In addition, adaptive FEC is the only technique that can maintain a specified level of reliability and also minimize delay as network conditions change. Our study shows that groupware AFEC can be a useful tool for improving the real-world performance and usability of real-time groupware.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: *CSCW*

General Terms

Performance, Design, Reliability, Human Factors.

Keywords

Synchronous groupware, latency, reliability, adaptive FEC

1. INTRODUCTION

Real-time distributed groupware often sends several different kinds of messages, each with different Quality of Service (QoS) requirements. For example, awareness messages like telepointer updates need low latency, but do not need to be completely reliable; transactional messages like object creation need reliability guarantees, but may not make strict latency demands. In addition, some message types can have different requirements depending on context, such as when telepointers are used for communicative gestures (where more messages will be needed to show smooth movement) instead of for general group awareness (where low-granularity positional information is often sufficient).

The QoS requirements of real-time groupware must be met either by infrastructural capabilities of networks or by application-level networking techniques that are controlled by the groupware developer. The dramatic improvements to Internet availability and pervasiveness over the past several years suggest that groupware's QoS requirements can be easily met by standard protocols and techniques, but experience shows that this is not always the case.

There are two main problems. First, Internet networks are not always able to provide the quality of service that is needed for real-time interaction through groupware. Although some network connections show latencies less than 100ms with near-zero loss, there are still many settings where both delay and loss are common. For example, problems can be caused when connecting from one broadband network to another, when using wireless or cellular networks, when connecting across inter-continental links, when network traffic is heavy, or when working in rural or remote regions with less-developed infrastructure [1,3,20,22,26]. As a result, users – even in North America or Europe – may frequently experience loss rates above 5% and delays above 250ms, with higher spikes not uncommon (e.g., [1,20]). In addition, the loss and latency that users experience are higher than published network statistics (e.g., network statistics do not include delays introduced in other parts of the connection such as input controllers or display hardware, and do not count loss episodes such as packets that are out of order or too late to be useful).

Second, the network protocols that groupware developers use most commonly (TCP in particular) are poorly suited to real-time communication in real-world networks. Loss is a particular problem if groupware uses TCP, since the in-order guarantee of the protocol means that any lost packet holds up all others until retransmission completes. Many groupware messages do not need in-order delivery (e.g., telepointer messages), but need low latency, and thus interaction can suffer greatly. As a result, game developers and groupware researchers have moved to UDP-based transport for real-time groupware. UDP has many advantages for real-time communication, but also provides no reliability guarantees – and in high loss environments, this lack can be just as problematic as the limitations of TCP. Researchers have developed reliable versions of UDP (e.g., [13]) but these are not tuned to the needs of groupware – for example, they often provide 100% reliability and often force in-order delivery, when groupware messages have varying reliability and ordering requirements, and would benefit from a more flexible approach.

Forward Error Correction (FEC) [2,19,22] is a technique well known in the networking world that could provide this flexibility, and could form the basis for a groupware-centered application-level reliability scheme. FEC duplicates messages in successive network packets, such that if one is lost, the message can still be retrieved from a later packet. FEC has the advantage of providing reliability without requiring acknowledgment or retransmission (which cause the latency problems of TCP-style reliability). In addition, adaptive versions of FEC (AFEC) can provide a wide variety of reliability rates that automatically tune themselves to current network conditions [2]: in low loss situations, the adaptive technique reduces redundancy which also reduces bandwidth requirements; as loss increases, redundancy is increased until a balance between reliability and latency is met.

GROUP'12, October 27-31, 2012, Sanibel Island, Florida, USA.

Adaptive Forward Error Correction is well suited to the needs of real-time groupware. However, although many researchers have studied FEC in the area of streaming media, and although it has been used in a few groupware projects (e.g., [5]), there has been no thorough analysis of the technique to establish its effects on the performance of real-time groupware.

In this paper, we carry out this investigation of forward error correction for groupware. We develop an AFEC technique designed around the reliability and latency requirements of real-time groupware, and carry out an evaluation to see the effects on groupware performance. Our study shows that the technique works very well at managing loss and latency in real-world network conditions, and is adaptable both to changes in network status and to changing demands at the application level. To make our AFEC technique widely available to groupware developers, we have built a reference implementation on top of the open-source Lidgren UDP library [14]. In this work, we provide strong evidence that AFEC is a powerful and practical technique for maintaining QoS requirements in real-time groupware; the benefits to user-level experience can substantially improve the usability of real-time collaboration at a distance.

2. BACKGROUND

Network issues for real-time systems have been studied in several domains, including distributed systems, networking, and CSCW. We review this research below, focusing on issues that are of greatest relevance to our investigation of FEC for groupware.

2.1 Quality of Service

Quality of Service (QoS) is a set of requirements for different aspects of computer networks that define performance levels for distributed systems. Many CSCW papers mention QoS and its importance (e.g. [17,18]), but there are few that discuss how to model or deliver QoS. This may be because QoS in real-time groupware is complex compared to other classes of applications, due to the wide variety of message types and the unconstrained nature of the tasks and interaction techniques [8,10].

QoS *models* include sets of QoS *characteristics* that are appropriate for a particular application type, as well as their associated QoS *parameters*. An early QoS model designed specifically for groupware is described by Mathur and Prakash [18], and includes four characteristics: latency, jitter, packet loss, and asynchrony. In the following sections, we look more closely at two of these characteristics: latency and loss. However, there are many other characteristics that could be considered for a complete QoS model of groupware, such as security, frequency, resolution, accuracy, precedence, and authenticity (which are described in the generic ISO/IEC QoS model [14]).

Managing QoS in groupware is also different from other applications because of the variable and dynamic nature of groupware sessions. Unlike systems that only transport streaming media (e.g., VoIP), groupware QoS techniques must adjust to the needs of different applications, scenarios, user preferences, and group dynamics. For example, Greenhalgh [10] describes a QoS manager for a virtual environment that allocates resources based both on application-specific settings and on a spatial awareness model to determine level of interest between participants. Although full QoS models are difficult for groupware, it is still possible to improve performance by considering individual QoS characteristics – such as delay and loss as discussed below.

2.2 Network Delay

Delay is a fact of life in real-world distributed applications because information must be transmitted across a network and processed at the other end before it can be displayed. There are two main types of delay – latency and jitter [22]. *Latency* is the time that elapses between an event and its display at another system, which results in a person's actions in the shared environment being seen after they actually occur. *Jitter* is the variation in latency due to changing network traffic conditions and processing loads; jitter reduces the smoothness of a remote user's displayed actions, such that motion (such as telepointer motion) looks halting and jerky [11].

Delays can have severe effects on collaboration – on coordination, communication, and understanding of the shared situation. Delay can make turn-taking difficult to negotiate, can hinder social protocols, and can cause inconsistencies that lead to confusion about the timing or simultaneity of key events [4,11].

There has been a large amount of research showing that network delay can negatively affects users of real-time groupware – but the effects of delay depend a great deal on the type of application, the degree of coupling between collaborators, and the temporal granularity of the interaction. For example, a study of a Pong-style game found that users did not seem to perceive latencies of less than 150ms, but that performance was affected with 500ms latency [30]; in contrast, studies of first-person shooters have found that players are sensitive to latencies below 100ms [26]. Studies of tightly-coupled coordinated interaction (where people must act based on what another person is doing) have also shown negative effects of latencies at or above 100ms [28]. Jitter has also been shown to have effects on group interaction, particularly on people's ability to predict others' movement [11]; but again, the effects of jitter are dependent on the environment and the interaction, and other studies show that these delays do not always affect users' perceptions in game environments [26].

Researchers have also considered numerous strategies for reducing or hiding network delay, such as dead-reckoning or local lag (e.g., [28]). These techniques are vitally important in an overall view of groupware QoS, but are not required for our work on reliability (although delay-compensation techniques can be integrated with the techniques proposed here).

2.3 Message Loss and Reliability

A second reality of distributed groupware is that not all messages can be successfully delivered to receivers. There are two main reasons for these delivery failures – packet loss and loss caused by out-of-order or late delivery.

Packet loss occurs whenever network packets sent by a distributed system fail to reach their destination, and is caused by a variety of problems including signal degradation, interference, congestion at routers (i.e., buffer overflows), or network errors [23]; there can even be losses induced by communication protocols themselves (e.g., some versions of the 802.11 protocol [25]).

Order-based loss occurs when a message is unusable at the receiver because it is delivered out of order, or too late. For example, telepointer messages that are delivered either late or out of order are no longer useful for showing a person's current location (although they may still be used to show traces [12]).

There are two main ways to describe loss. First, the overall loss rate (e.g., percentage of packets lost in a given time period) describes general behavior over time. Second, and more importantly for groupware, the *burst characteristics* describe the distribution of loss at the local level. Burst loss means losing more than one packet in a row, and can be described in terms of the *burst length* (the number of consecutive packets lost), and the *burst frequency* (the rate at which bursts occur) [23].

Although many networks have near-zero loss on average, there are still many settings and situations where loss can be a problem. For example, an informal five-day review of the Internet Health Report (www.internetpulse.net), which reports statistics for main North American Internet providers, showed hourly average loss rates above 2% 104 times, and above 5% 45 times (rolling hourly averages, sampled every ten minutes). Even low loss rates can be problematic if losses occur in bursts, a phenomenon that is common when traffic levels are high [1].

Losses on wireless networks (cellular networks and wireless LANs) can be much higher, up to and even higher than 20% on average for long-distance WiFi networks [20]. Overall, these situations show that techniques are needed to control the reliability of messages in distributed systems.

2.4 Techniques for Message Reliability

Reliability techniques are used to counteract ordering and loss problems that occur while sending information over the network. There are two main approaches: protocol-based reliability, and application-level techniques.

2.4.1 Protocol-based reliability

Some communications protocols (e.g., TCP/IP) build reliability into their specifications, and groupware can in some cases maintain QoS simply by selecting an appropriate protocol. However, this approach has problems in that protocols are very general tools that cannot be adequately tuned to the needs of real-time groupware.

The TCP/IP protocol shows both the value and the limitations of the infrastructural approach to reliability. TCP is simple to use for programmers, and guarantees delivery (and in-order sequencing) of messages. However, guaranteed in-order delivery does not match the QoS requirements of many types of groupware messages, and the considerable resources used for acknowledging and retransmitting lost information leads to severe performance problems when TCP is used on real-world networks [5,7].

The other network protocol used for real-time groupware is UDP/IP. Since UDP provides no reliability or ordering guarantees, custom protocols are often built on top of UDP to deliver various QoS levels [7]. Application-level protocols are discussed in the next section, but here we consider RTP, a well-established protocol that is frequently used for sending real-time voice and video information. Some researchers have used RTP for real-time groupware applications (e.g. [8]), and there are RTP payload formats for common groupware message types including text messages and telepointers. The performance of RTP for groupware has not been tested, however, and some analyses indicate that RTP is not well suited to delivering the wide range of QoS requirements exhibited by real-time groupware [24].

There is an overall trend (e.g., as seen in networked game libraries [7]) to avoid protocol-based solutions to QoS requirements. One reason is that protocols work only at the packet level, and there are many advantages to working at the level of messages instead. For example, if two message types have differing QoS requirements, these cannot be handled appropriately by a single protocol. The reliability needs of real-time groupware are most likely to be met by a message-level approach.

2.4.2 Application-level reliability

There are three main techniques for maintaining reliability levels in an application-level protocol based on UDP/IP: retransmission, path diversity, and forward error correction.

Retransmission techniques resend data that is known to be lost. There are two main ways to do this – by acknowledging all information when it arrives (ACK-based) or by requesting information when loss is detected (NACK-based) [25]. *ACK-based* error correction requires all received packets to be acknowledged; *NACK-based* techniques do not have this overhead, but require message sequence numbers in order to detect loss at the receiver. A further requirement is a continuous flow of information, since loss can only be detected after subsequent packets have arrived (it is also possible to send termination packets to indicate the end of a transmission).

Path diversity schemes sends information redundantly over two separate network routes, which can improve reliability because different network routes have uncorrelated loss patterns [16]. This technique has been shown to be successful for services such as VoIP and real-time streaming video [16]. However, current networks do not support the ability to define a network route, and so this technique is not yet feasible in most real-world situations.

Forward error correction (FEC) improves reliability by repeating information in subsequent network packets [2,19]. In the event that a packet is lost, the information can be recovered when subsequent packets arrive (see Figure 1). No retransmission is required, but additional bandwidth is used to deliver the same total amount of information. In addition, this method is only partially reliable, since burst losses that are larger than the amount of redundancy can still lead to lost information. FEC is typically parameterized with the number of repetitions used; for example, FEC-3 duplicates the previous three messages in each new packet. There are several variants of FEC that have been investigated, and the technique is used successfully in many streaming-media situations [19]; as described below, FEC is also well suited to the needs of real-time groupware.

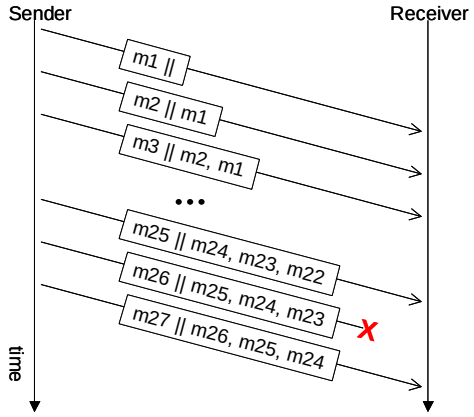

Figure 1: FEC-3 example. The last three messages are repeated in subsequent packets, allowing lost information (e.g., message 26) to be recovered without retransmission.

Adaptive FEC (AFEC) dynamically adjusts the amount of redundancy in order to meet QoS requirements while attempting to minimize the network cost of sending repeated information [2]. Adaptive FEC requires that the network conditions and the current reliability levels be monitored, and that an adaptation service continually adjust redundancy levels in response to network conditions or changing requirements.

Interleaved FEC is another variant of the basic FEC technique that disperses contiguous information among several packets by reordering portions of the messages [25]. For example, a stream that sends 20 messages per second at a packet rate of 5 packets

per second (4 messages per packet) could be interleaved by sending messages 1, 5, 9, and 13 in packet 1, messages 2, 6, 10, and 14 in packet 2, and so on. In the event of a single packet loss, less contiguous information would be lost, which could result in a smoother reconstructed stream at the receiver. However, this comes at the cost of added latency, since the information cannot be reordered and played out at the receiver until all interleaved information is received.

3. AFEC FOR REAL-TIME GROUPWARE

In developing an AFEC technique for real-time groupware, we need to consider three issues: the reliability requirements of real-time groupware, the characteristics of groupware messages that could affect the design of the technique, and the specifics of the technique itself. We initially considered applying an existing AFEC technique from VoIP research, but found that there are substantial differences between groupware and the streaming-media applications that have used AFEC in the past.

3.1 Reliability requirements of groupware

Groupware applications have reliability requirements that vary depending on the type of groupware data being sent. For instance, telepointer updates do not have to be completely reliable since a small number of lost positions will not inhibit collaboration or cause shared data to be incorrect. However, adding a new object to a diagram must be a reliable operation since a lost message would make model layer data inconsistent. For this reason, we differentiate between *transactions*, which require guaranteed delivery, and *awareness messages*, which can have reliability requirements, but do not require guaranteed delivery.

Transactions occur infrequently in groupware applications compared with awareness messages. Additionally, transactions can be far less sensitive to delay, especially if awareness messages are sent to indicate that a transaction is in progress. Considering the requirement for guaranteed delivery and lower delay sensitivity, as well as the infrequency of transactions, an ACK-based protocol such as TCP seems suitable for transactions, although alternatives may be appropriate as well.

Awareness messages are sent very frequently, do not require guaranteed delivery, and require low delay. ACK-based protocols are not efficient in this situation, and plain UDP is not a suitable approach either because of unpredictable reliability, which is not acceptable for certain awareness techniques. Groupware awareness messages require efficient transmission that minimizes delay while guaranteeing a specific level of reliability.

Table 1. Message types and QoS requirements for a shared-whiteboard groupware system

Message Type	Update Rate (min..max)	Latency	Reliability
AddObject	on event	250ms	100%
DeleteObject	on event	500ms	100%
MoveObject (intermediate)	5/sec..25/sec	100ms	80%
MoveObject (final)	on event	500ms	100%
ResizeObject (intermediate)	5/sec..25/sec	100ms	80%
ResizeObject (final)	on event	500ms	100%
ModifyText (intermediate)	1/sec..5/sec	100ms	95%
ModifyText (final)	on event	500ms	100%
MoveViewport	2/sec..5/sec	250ms	80%
Telepointer	10/sec..25/sec	100ms	80%
Telepointer (during voice)	20/sec..30/sec	100ms	90%

As an example, we designed a hypothetical shared whiteboard system, and identified message types and QoS parameters for those types (see Table 1). Requirements were determined by considering how each of the event types would be realized in the application and the effects of each of the characteristics on the user and on the application. Although these values are guided by our own intuition and experience (there is no formal approach to assigning QoS requirements for groupware), the diversity of values indicates a need for flexibility in handling reliability.

3.2 Characteristics of groupware messages

In addition to the varying reliability and latency requirements described above, real-time groupware has several characteristics that affect the design of a forward-error-correction technique.

Small and atomic messages. Groupware messages – particularly awareness messages – are small compared to the capacity of a network packet. For example, telepointer messages use 10-25 bytes when encoded efficiently [5], and other types of awareness messages (e.g., object or viewport moves) will be similar. This implies that more redundancy can be added to groupware packets, potentially allowing recovery from longer burst losses. In addition, since groupware messages often consist of state updates, event objects, or remote procedure calls, they must be sent in their complete format. This means that two techniques used in multimedia systems – sending partial data or sending a lower resolution version of the data – cannot be generally used for real-time groupware. Therefore, an encoding scheme for groupware should only include complete messages, a requirement can be met due to the small size characteristic.

Multiple message types. As seen in Table 1, several different types of groupware messages can be sent during the same session, each with different QoS requirements for packet loss, update rate, and latency. Different message types with different QoS requirements means that a groupware FEC scheme must handle the tracking and management of each message type separately. QoS performance and loss must be separately monitored for each message type, and the redundancy encoding scheme must make type-by-type decisions about which messages to include as redundant information to meet QoS requirements.

Bursty traffic. Groupware traffic is typically bursty, meaning that messages are clustered together with pauses in between. As an example, Figure 2 shows an activity trace of a shared whiteboard system [5] – in which telepointers, object manipulations, and text editing events all occur in clusters separated by inactive periods.

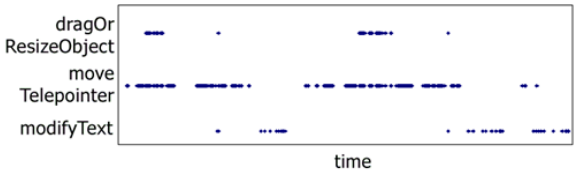

Figure 2: Occurrences of different groupware event types over time, for one user [5].

The clustering of awareness messages in groupware reflects the way that people work: telepointer motion, for example, is most often a series of moves with stops in between, rather than consistent and continuous movement. Note that we assume that awareness messages are event-driven and so are only sent out when something is happening. For a groupware FEC technique, clustering implies that when losses occur at the end of a cluster (e.g. just before a mouse move stops), messages cannot be recovered until more messages are sent. To deal with this, senders must detect the end of a message cluster and send packets containing only redundant messages until the recovery

requirements are met. A similar requirement is mentioned in IETF RFC 2793 when using redundancy in text messaging applications.

Individual and contextual differences. Different people and different tasks can produce very different message patterns within the same groupware system. Different people take on different roles at different times (e.g., leader vs. worker, or active vs. passive), and people also show individual differences in the way that they work. For example, Figure 3 shows a summary of message types generated by four users carrying out a single shared task in a shared workspace [5]: two users show relatively balanced totals for four different message types, and two users differ dramatically. These dynamic differences mean that a groupware FEC technique must be able to respond to application-level and user-level changes to QoS requirements. For example, user preferences or application monitoring might determine that a user who gestures frequently to communicate should have different QoS requirements for telepointer messages.

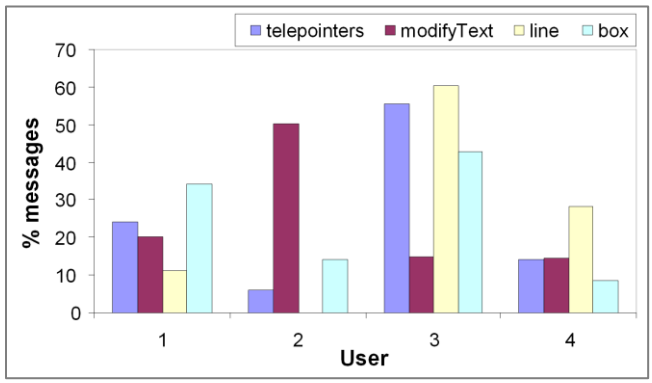

Figure 3: Message types generated by four users during a shared diagramming task [5].

3.3 The GW-AFEC Technique

We designed an AFEC technique that is tailored to meet the specific requirements of real-time groupware. The technique consists of an encoding scheme, an adaptation logic, and a code selection algorithm (which determines the number of redundant messages to be included in an outgoing packet). Our technique is based on the assumptions that messages are small, that messages must be sent in their entirety, and that the receiver knows the minimum and maximum reliability requirements for each message type. We use the term message error rate (MER) to refer to the percentage of messages that are never received. This is different from the packet error rate (PER), which is the percentage of packets lost. Note that with reliable protocols, MER is always zero and for unreliable protocols, MER is equal to PER. When using error correction, PER is greater than or equal to MER because messages contained in lost packets can be recovered.

One might ask why we would ever set a non-zero minimum for message error rate; the answer is that for information that can tolerate some loss without any problems for the users, the extra bandwidth needed to provide further reliability could be better allocated to another channel that can make better use of it (e.g., sending accompanying video at higher resolution).

The encoding scheme that we used is simple: redundant messages are included in their entirety and in their original form, rather than being distributed over several packets or sent at lower resolution. This approach works because of the small size of awareness messages. Several redundant messages can be added to each packet, allowing full recovery of lost messages, even from small burst losses. The total number of redundant messages still varies, however, depending on the network bandwidth, the actual message size, and the message frequency.

Sending entire messages allows full recovery from a single packet. For example, adding just one redundant message allows full recovery from any single lost packet. If two packets are never lost in a row, this scheme would provide full reliability. In general, however, accommodating burst losses means that a scheme must include as many redundant messages in each packet as the maximum burst length, in order to prevent any message loss. In cases where this cannot be achieved, the number of messages lost during a burst is the number of packets lost during the burst minus the number of redundant messages in a packet.

Since it is difficult to predict loss occurrence and burst length [1], it is not feasible to determine the number of redundant messages required to meet MER requirements in advance. Although MER can be reduced significantly using a fixed amount of redundancy, non-adaptive schemes cannot guarantee reliability. An adaptive approach is required to meet MER requirements while keeping the amount of redundancy to a minimum.

The need to accommodate MER requirements for a variety of message types also makes the adaptive logic more complex. In our technique, the receiver monitors each message type separately to ensure that MER requirements are being met, and tells the sender when to add or remove redundancy. The sender uses a code selection algorithm that considers each message type separately to meet the redundancy requirements. Since there can be many participants in a groupware system, MER is monitored separately for each user by the receiver, and the redundancy requirements are tracked separately for each user by the sender.

In the next two sections we provide more details on the elements of groupware AFEC that are required in the receiver and the sender of a message (see Figure 4).

3.3.1 Receiver-side responsibilities

The receiver is responsible for monitoring MERs to ensure that they stay between the minimum and maximum values set out for each message type from each user. Incoming packets consist of one current message and some number of redundant messages, all from a single user. The messages in a packet are processed in reverse order from oldest to newest (thus, redundant messages are processed first). It is important to note that this guarantees in-order delivery, since late out-of-order packets will be discarded because the messages they contain will have been recovered already. The receiver checks for message loss using message indexes (described below); if loss has occurred, the receiver increments the loss counter for that message type. Any message that has not been seen before (including recovered and new messages) is then processed as required.

Message loss detection is performed using message indexes. The highest received index is stored for each message type, and if the index of a message exceeds this value by more than one, the number of lost messages is reported and the MER for that message type is updated. To be able to record MER correctly for each message type, we need to know what type of message was lost. Therefore, we specify type within the index using a numeric message type code that precedes the index number.

Current MER is calculated using the previous 1000 messages only, rather than all of the messages during the life of the session. This is necessary to allow the system to react to changes in network conditions and to ensure that adjustments quickly result

in different behavior. Otherwise, average rates would be spread out over the life of the session, resulting in reduced responsiveness as the number of messages in the session increases. The rolling average could also be accomplished using a time-based window; however, since the message frequency is irregular, a time-based approach could result in too small of a sample size to make accurate changes to the code set size (the number of redundant messages in a packet).

The receiver is responsible for deciding when to increase or decrease the amount of redundant information being sent. If the MER exceeds the maximum MER, the code set size must be increased so that more messages can be recovered. When the MER is below the minimum, the code set size must be decreased to avoid unnecessary redundancy (since it increases traffic without benefit). These control actions are accomplished with a module that periodically checks each message type's MER. If any of the MERs are above the maximum, a negative acknowledgement (NAK) is sent to the sender immediately, telling the sender to increase the amount of redundancy for the specified message type. If a MER is below its minimum value, the receiver sends a decrease acknowledgement (DEC) message, which tells the sender to reduce the redundancy for the specified message type.

Since groupware awareness messages are often clustered, code set size is only be updated when there is significant activity. With low activity, there is a risk of overcompensation because the code selector runs in a timed thread and could update the system several times without receiving new information.

3.3.2 Sender-side responsibilities

The sender is responsible for meeting the redundancy requirements as requested by the receiver. The code selection algorithm works to ensure that the correct number of redundant messages is included in each packet as requested by the receiver. The sender keeps a history buffer of each message type and knows how many times each message has been sent. When loading messages into a packet, the encoder starts from the most recent message and moves back through the history list. Only messages whose requirements have not yet been met are included.

The sender keeps a tally of the packet size and ensures that the amount of redundancy added does not exceed the path maximum transfer unit (path MTU) for the network route. The path MTU is the largest packet size that is forwarded by a router [23]. If the path MTU is exceeded, the packet will be divided into smaller packets, which adds delay. Therefore, the path MTU must be discovered by the system and should not be exceeded. In the case that adding redundancy will exceed the path MTU, the sender sends as many messages as it can without exceeding the path MTU and sends a notification to the receiver that the QoS requirements cannot be met and should be lowered. Adjustment of requirements can be handled automatically by the application or by the user through a dialog.

Finally, since groupware messages are often strongly clustered, times in which there are no messages to be sent are detected and a few extra messages are sent from the history buffer, in order to meet redundancy requirements for the last few messages of the cluster. Without adding messages after a cluster, MER would appear to be higher for the last few messages in each burst.

It is important to note that messages recovered using FEC or AFEC arrive at the same time as the newest message, which creates a similar effect to that of network jitter. Interface-level solutions are required to deal appropriately with this late information. One method is to use client-side buffering, which adds additional latency. Another method that is suitable for groupware is to use traces, a technique that visualizes recent movement, improving accuracy without adding latency [12].

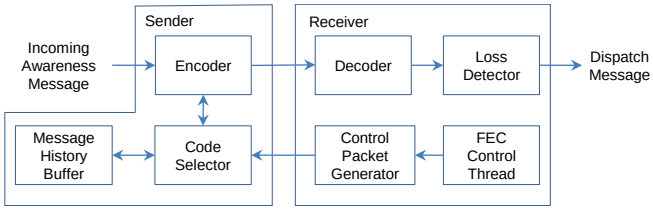

Figure 4: AFEC operation: incoming and redundant messages are combined based on the Code Selector's decisions; encoded messages are sent via UDP and decoded at the Receiver, which detects and reports loss; a timed FEC Control Thread monitors MER and adjusts the amount of redundancy.

3.3.3 Implementation

The GW-AFEC technique has been implemented both in Java and in C# using the Lidgren UDP library [15]. The system includes all of the modules shown in Figure 3, and can easily be adapted to work with many groupware architectures and event schemes. Our implementation is available at http://hci.usask.ca/gw-afec/.

4. EVALUATION OF GW-AFEC

We ran a set of experiments to see how the GW-AFEC technique would compare with TCP, plain UDP, and non-adaptive fixed-length FEC schemes under a variety of network conditions. Network conditions were simulated on a LAN using a software-based network-disabling emulator [21]. For each experiment, we measured MER and latency for each message. Our goals were to determine the following:

- How a guaranteed protocol like TCP compares with UDP-based schemes in lossy conditions;
- How FEC performance compares to plain UDP for a variety of code set sizes and bandwidth constraints;
- If and when non-adaptive FEC performs poorly;
- How AFEC compares with non-adaptive FEC;
- If and when AFEC performs poorly.

The test application was a simple telepointer application with a movement trail. We used this common awareness technique to ensure that the message patterns were similar to those found in real world groupware applications. An input message trace was recorded by moving a mouse manually in a manner that simulated natural activity with a whiteboard. This simulated the clusters of messages that happen in a groupware application. The update frequency for the telepointer was fixed at a maximum rate of 30 updates/second for all tests.

Simulations using the input trace were then run with two clients on the same machine, but with messages sent through a server on a different machine on the LAN. The network emulator ran on the server machine, simulating a variety of network conditions between the two clients. This setup enabled accurate latency measurements using the system clock on the client machine.

Loss rates, loss patterns, and amount of bandwidth were specified using the emulator to simulate different network conditions:

- *Loss rates*. We used three loss rates: 0%, 10%, and 20%.
- *Loss patterns*. Two different loss patterns were used: random and burst. The random pattern loses packets randomly based on a percentage chance of loss. We ran random loss tests at 0%, 10%, and 20% loss. For burst loss, two parameters determined the loss rate: the number of packets that would be

lost in a burst and the probability of a burst loss occurrence. Burst loss tests were run with 1-5 packets being lost with a 4% chance of starting on any packet (10% overall loss), 1-5 packets being lost with an 8% chance (20% overall), 1-10 packets being lost with a 2% chance (10% overall), and 1-10 packets being lost with a 4% chance (20% overall).

- *Bandwidth*. Each experiment was run using two levels of available bandwidth: 56Kbps and 256Kbps. Although total bandwidth will generally be higher than these levels, traffic patterns in real-world networks mean that the amount of available bandwidth is often restricted. We chose these levels as two types of resource-constrained environment.

For non-adaptive FEC, we ran tests with a variety of code set sizes (i.e., number of redundant messages) between 2 and 7. Since different code set sizes lead to different performance characteristics, we consider each as an independent technique, and will refer to them as 'FEC-n' where n is the code set size. (Note that in Figures 5 and 7, we show FEC-3, which was the best-performing of the non-adaptive techniques).

The experiments with AFEC were run with a target MER range of 5% minimum and 6% maximum. In a telepointer example, this means that we want to ensure that at least 94% of position messages arrive, but that more 95% is not needed for the type of interaction being supported. For each experiment, we measured message latency and MER.

4.1 Results

The results from our experiments are organized by the goals stated above – how different protocols perform in realistic conditions, the advantages and disadvantages of the basic FEC approach, and the behavior and performance of groupware AFEC.

4.1.1 Comparison in a realistic network situation

Our first investigation compares all protocols in terms of message latency and message error rate. Figure 5 shows average latency in a 56Kbps channel with 10% random loss for TCP, UDP, FEC-3 (i.e. FEC with code set size of 3), and groupware AFEC. Figure 7 shows MER for the same set of conditions. The most obvious result in Figure 5 is that when networks are experiencing loss, TCP's acknowledgment and retransmission policy lead to large latency. TCP latency is larger even in networks with no loss, but as loss increases, TCP quickly becomes unusable. In addition, the high variance in latency with TCP (Figure 6) makes it difficult for users to adapt to the delay.

In contrast, all of the schemes based on UDP maintain a low average latency. Differences among these three schemes can be attributed to packet size and to the resultant traffic level, since higher traffic in a limited channel generally corresponds to higher latency. UDP has the smallest packet size (equivalent to FEC-1) and therefore generates the least traffic; FEC-3 has the largest packet in this scenario, and so has slightly higher latency.

Results for message error rate (Figure 7) also show substantial differences between the protocols. Since TCP is a guaranteed reliable protocol, MER is always 0%. Since UDP provides no reliability control, its MER will always be approximately equal to the packet loss rate (here 10%). Groupware AFEC has a MER of 5%, which is within the bounds of the target MER of 5-6%. FEC-3 in this case was highly reliable – in fact too reliable, in that it used bandwidth that could have been better allocated to a different information channel (such as a video stream).

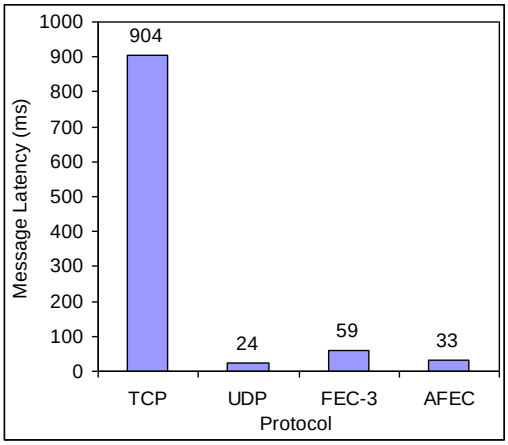

Figure 5: Average latency, by protocol, 56Kbps available bandwidth, 10% random loss.

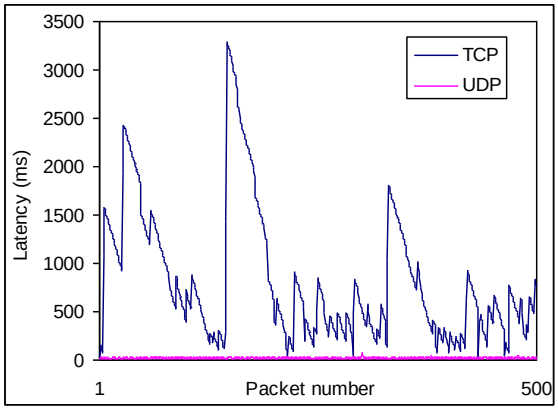

Figure 6: Typical variation in TCP and UDP latency, 56Kbps available bandwidth, 10% random loss.

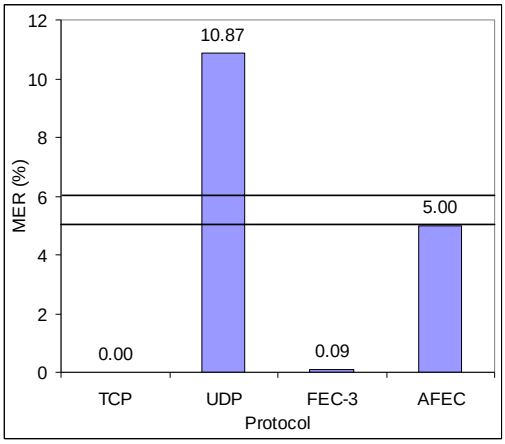

Figure 7: MER by protocol, 56Kbps available bandwidth, 10% random loss. The target bounds for MER (5%-6%) are marked with lines.

4.1.2 Performance of non-adaptive FEC

We tested non-adaptive FEC with several different code set sizes. In all tests, reliability was much higher than plain UDP, and in some experiments non-adaptive FEC showed the best performance for both MER and latency. However, these cases occurred only when the code set size happened to be an

appropriate choice for the network conditions. When the code set size was not appropriate for conditions, then non-adaptive FEC showed either higher latency or an MER that was below the target range, indicating excess bandwidth usage.

One variable that greatly affected non-adaptive FEC was loss pattern. Random loss experiments resulted in very low MERs. Tests with burst loss, however, resulted in much higher MER values. The poor performance occurred because messages are lost whenever the size of the burst exceeds the code set size.

In general, although non-adaptive FEC provides an enormous improvement over plain UDP, an inappropriate choice of code set size can lead to reduced performance. This can be seen in Figure 7, where, the latency and MER of FEC with various code set sizes is compared to AFEC. Although different code set sizes lead to different performance extremes, only a few values provide a balance between latency and error rate.

4.1.3 Performance and characteristics of AFEC

The performance of AFEC was always equal to or better than that of non-adaptive FEC, as long as it was possible to meet the target MER range without exceeding the available bandwidth. AFEC shows optimal performance over time because it always moves towards the minimum code set size needed to meet the target MER range. In cases where non-adaptive FEC exceeded the reliability requirements (i.e. had a larger code set size), AFEC showed lower latency due to its smaller packet size. AFEC also selected a higher code set size to meet the target MER range in cases where FEC did not meet reliability requirements. The MER for AFEC varied between 4% and 7%, but was most often within the target range of 5% and 6%. In high burst loss conditions, AFEC was still able to meet the target MER range with low latency, as long as the available bandwidth was sufficient.

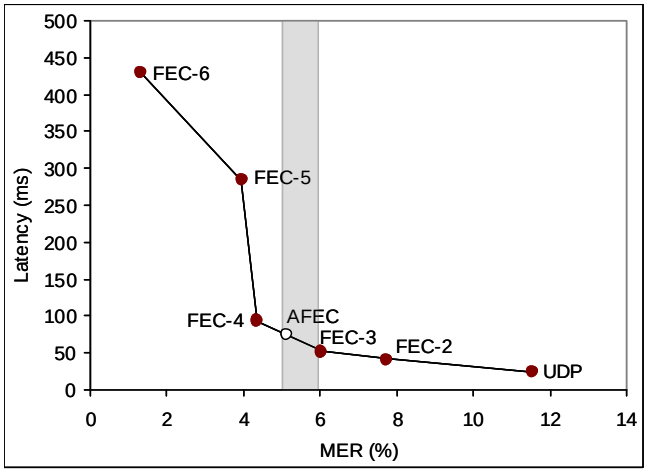

Figure 8. Average latency and MER for non-adaptive FEC with various code set sizes, and AFEC (white circle), on a 56Kbps channel with burst loss (burst size of 1-10 and 2% probability of occurrence).

Figures 9 and 10 illustrate these differences and show how AFEC works. Figure 9 shows latency of various techniques as loss increases from zero to 20%. Because AFEC varies its code set size, it can perform as fast as UDP when loss is low. As loss increases, AFEC's increasing packet size leads to higher latency; however, this is the minimum latency possible when maintaining the QoS requirements for MER.

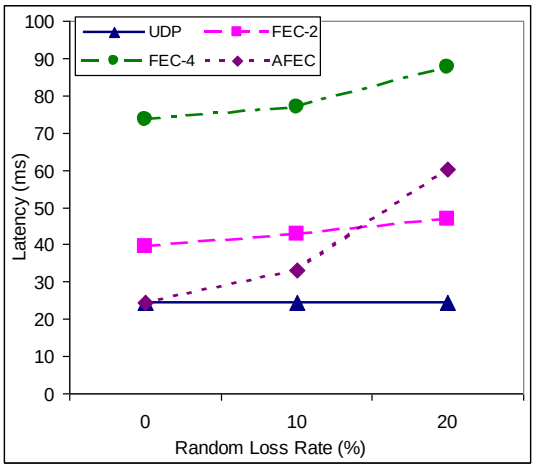

Figure 9. Average latency of different protocols at three loss rates. Note that AFEC latency increases as a result of increasing redundancy to maintain reliability.

The behaviour of AFEC is illustrated in the packet trace shown in Figure 10. At the start of the trace, AFEC has a code set size that is slightly too large, resulting in a gradual decline in MER past the specified minimum. Once below 5%, AFEC reduces code set size by one (line A in Figure 10); since this reduces packet size, latency decreases. The reduction in code set size is not enough to prevent another decline past the minimum (line B), so AFEC reduces again. This results in low latency, but also in a code set that is slightly too small for network conditions, so MER climbs rapidly. When the MER is observed outside the maximum, code set size is increased (line C); this stops the increase in MER, but does not reduce it below the maximum, so another code-set increase is made (line D), also increasing latency.

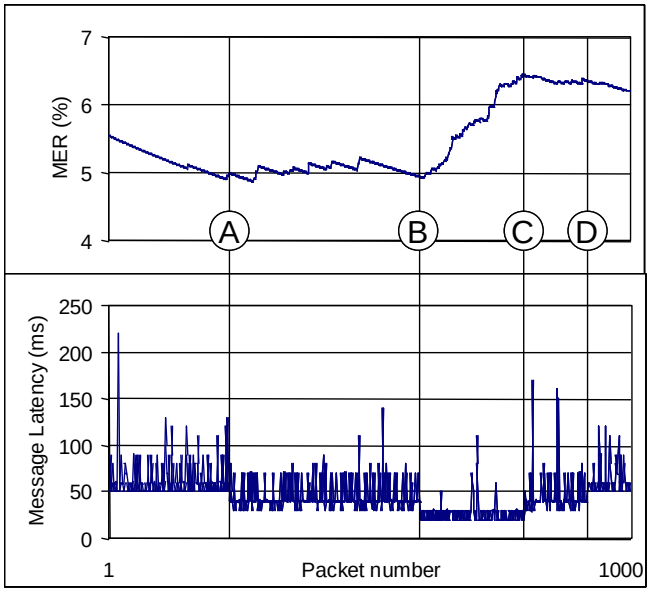

Figure 10: MER (upper) and per-message latency (lower) for 1000 AFEC packets, on a 56Kbps channel with 20% random loss. Vertical lines indicate points at which AFEC changed its code set size to stay within MER limits.

Note that the latency spikes in Figure 10 result from the latency of recovered messages, not from packet latency. Whenever messages are recovered, additional latency results from the time between the first send of the message and its eventual receipt as redundant

information. Thus, message latency in AFEC is always slightly higher than packet latency.

Finally, our experiments showed certain conditions where the current AFEC technique must be paired with other techniques to be used successfully. We did not use adaptive rate control in our experiments, so certain situations involving high burst loss and low bandwidth resulted in high amounts of latency. This problem arose when AFEC attempted to increase its code set size past the bandwidth limits in order to meet MER requirements. However, when used with rate control, this problem can be avoided.

5. DISCUSSION

These experiments compared the effectiveness of TCP, UDP, FEC, and AFEC for sending real-time awareness data under a variety of network conditions. The following conclusions can be drawn from our results:

- TCP is not suitable for sending real-time interactive messages over lossy networks due to very high latency;
- FEC produces substantial reliability increases over UDP alone without adding much latency in most cases;
- Optimal code set size is impossible to determine beforehand in most cases because of unpredictable network conditions;
- AFEC can meet a predefined MER range while minimizing the amount of latency, given a fixed send rate;
- When it is not possible to meet the level of reliability without exceeding bandwidth requirements, the system must either decrease send rate, decrease MER requirements, or accept higher latency.

Our experiments reinforce earlier indications that TCP is not suitable for sending real-time interactive messages where guaranteed delivery is not required. The high amount of latency and jitter of TCP do not meet the requirements for real-time data under lossy conditions. These results help to confirm that TCP should not be used for sending real-time awareness information.

Non-adaptive FEC is simple to implement and provides large reliability benefits over plain UDP. In general, a small code set size (e.g., 2) can be used to improve reliability considerably without adding substantial latency. However, a small code set size may not always provide the desired level of reliability, so under certain conditions, larger code set sizes may be better choices. Several factors affect the amount of latency that is added, and the level of reliability attainable using FEC.

The amount of latency added by FEC depends on the available network bandwidth, message size, message frequency, number of users, and code set size. In situations where bandwidth is plentiful, large code set sizes add a trivial amount of latency. However, low available-bandwidth conditions can cause the problem described above, where code set size cannot be supported by the network. Therefore, when clients have large amounts of bandwidth, a larger code set size is a good choice, but when bandwidth is low, the code set size must be set at a level that does not exceed resources.

The level of reliability provided by FEC depends on the loss type, the loss rate, and the code set size. A larger code set size always results in the same or better reliability because it allows a longer sequence of lost messages to be recovered. Higher loss rates always result in the same or worse reliability because the frequency of losses is higher. Random losses where few packets in a row are lost result in high reliability from FEC, while longer bursts result in lower levels of reliability. Therefore, when long, frequent burst losses are occurring, a larger code set size is desirable, but when only short bursts occur less frequently, a smaller code set size is sufficient.

The problem with non-adaptive FEC is that the application programmer must implement the code set at a fixed size, but an optimal choice depends on several unpredictable parameters: loss rate, loss pattern, and available bandwidth. If the programmer knew the network conditions beforehand, an optimal code set size could be selected. Since this is not possible, the application programmer is left with the problem of determining the optimal size. Therefore, the code set size must be selected conservatively and without certainty based on the cost-benefit analysis of reliability versus latency under unknown conditions.

AFEC improves on non-adaptive FEC by dynamically adapting its code set size to meet the MER requirements for each technique. This removes the uncertainty that results from choosing a fixed code set size for non-adaptive FEC. AFEC adjusts the code set size to the minimum that still maintains the desired MER range. In cases where non-adaptive FEC exceeds the required level of reliability, AFEC will always produce lower latency. When non-adaptive FEC does not meet the MER requirements, AFEC will automatically increase its code set size to meet the reliability requirements, as long as this does not exceed the path MTU.

The only remaining problem with AFEC is that it is sometimes not possible to meet the desired level of reliability at a certain message frequency. In these cases, either rate can be decreased or MER requirements can be lowered. The GW-AFEC module presented here does not include adaptive rate control, although this can be added in future work.

Overall, GW-AFEC is an effective technique for sending real-time interactive messages. AFEC should be considered as an addition to groupware toolkits, as it provides substantial benefits under a wide range of conditions, and because its complexity would be best abstracted away from application programmers.

6. FUTURE WORK

GW-AFEC is one of several application-level techniques that can help to improve groupware performance and usability on real-world networks. There are several future possibilities both for advances in this technique and in the general area of groupware networking. To develop better techniques for supporting real-time interaction in groupware, we need to gain a better understanding of the characteristics of groupware traffic and the QoS goals that we are trying to meet. This paper discusses a few basic characteristics of groupware traffic and important QoS characteristics. However, better knowledge of both network and QoS characteristics will facilitate development of more effective techniques that can be applied to groupware.

Once we have better knowledge of characteristics and QoS requirements, we can refine GW-AFEC and develop other techniques that take advantage of the specific characteristics and opportunities of real-time groupware. Some techniques have been considered in other work (e.g., compression [5] or latency compensation [28,29]); we are also planning to explore techniques such as adaptive concurrency policies, distributed load balancing, adaptive rate control, and receiver-driven layered multicast. The collection of networking techniques can eventually be brought together in a new real-time groupware toolkit.

7. CONCLUSION

Our experiments show that AFEC is an effective technique for sending real-time interactive messages because it can meet reliability requirements while minimizing latency under lossy

conditions. AFEC works by adaptively adjusting the amount of redundancy in packets so that lost messages can be recovered without requiring retransmission. Groupware messages are generally small, which allows several redundant messages to be added to each packet. This allows AFEC to recover lost messages, even when burst losses occur.

AFEC for groupware is different from AFEC for multimedia due to several key differences between the application types. Since groupware sends many different message types with different reliability requirements, AFEC must include monitoring and tracking support for multiple message type. Irregular bursts of messages that occur in groupware can be handled by detecting breaks between bursts and sending redundant information at the end of the bursts. Groupware's small messages and payload types made it appropriate to send complete messages rather than compressed portions of historic messages.

An experiment comparing several protocols in realistic lossy network conditions shows that non-adaptive FEC improves on the reliability of UDP, but code set size must be chosen carefully. In dynamic network conditions, however, this choice is impossible to make correctly, whereas GW-AFEC is able to move towards the optimal value in most cases. Our experiments also reinforce that TCP is unsuitable for sending real-time interactive messages, especially under lossy conditions. Our results show that GW-AFEC is a useful strategy for sending interactive real-time groupware messages, and that reducing delay while providing needed reliability can substantially improve groupware usability.

8. REFERENCES

1. Bolot, J-C End-to-end packet delay and loss behavior in the Internet, *ACM SIGCOMM Computer Communication Review*, 23, 4, 1993, 289–298.
2. Bolot, J-C, Fosse-Parisis, S., Towsley, D. Adaptive FEC-Based error control for Internet Telephony, *Proc. Infocom 1999*, 1453 - 1460.
3. Chebrolu, K., Raman, B., and Sen, S., Long-distance 802.11b links: performance measurements and experience. *Proc. ACM MobiCom 2006*, 74-85.
4. Claypool, M., and Claypool, K., Latency and player actions in online games. *Comm. ACM*, 49, 11, 2006, 40-45
5. Dyck, J., and Gutwin, C., and Makaroff, D., *Using Behaviour Characteristics to Improve Groupware Performance*, Technical Report HCI-03-01, Univ. of Saskatchewan, 2003.
6. Dyck, J., Gutwin, C., Subramanian, S., and Fedak, C. High-Performance Telepointers. *Proc. CSCW 2004*, 172-181.
7. Dyck, J., Gutwin, C., Graham, N., and Pinelle, D., Beyond the LAN: Techniques for Improving Groupware Performance from Networked Games, *Proc. Group 2007*, 291-300.
8. Frécon, E., Greenhalgh, C., Stenius, M., The DiveBone - an application-level network architecture for Internet-based CVEs. *Proc. ACM VRST 1999*, 58-65.
9. Gracanin, D., Zhou, Y., and DaSilva, L. Quality of Service for Networked Virtual Environments. *IEEE Communications Magazine*, April 2004, 42-48.
10. Greenhalgh, C., Benford, S., Craven, M. Patterns of network and user activity in an inhabited television event. *Proc. ACM VRST 1999*, 35-50.
11. Gutwin, C. Effects of Network Delay on Group Work in Shared Workspaces. *Proc. ECSCW 2001*, 299-318.
12. Gutwin, C. Traces: Visualizing the Immediate Past to Support Group Interaction. *Proc. GI 2002*, 43-50.
13. IETF Network Working Group, RFC 1151, *Version 2 of the Reliable Data Protocol*, 1990, tools.ietf.org/html/rfc1151.
14. ISO/IEC. *Quality of Service: Framework*. ISO/IEC 13236, 1998. www.iso.org.
15. Lidgren UDP Library. code.google.com/p/lidgren-network-gen3/, 2011.
16. Liang, Y., Steinbach, E., Girod, B. Real-time voice communication over the internet using packet path diversity. *Proc. ACM Multimedia 2001*, 431-440.
17. Marsic, I. Real-Time Collaboration in Heterogeneous Computing Environments. *Proc. ITCC 2000*, 222-227.
18. Mathur, A., Prakash, A. *A Protocol Composition-Based Approach to QoS Control in Collaboration Systems*. Technical Report CSE-TR-27495, U. of Michigan, 1995.
19. Nafaa, A., Taleb, T., and Murphy, L., Forward Error Correction Strategies for Media Streaming over Wireless Networks, *IEEE Communications*, Dec. 2007, 72-79.
20. Sheth, A., Nedevschi, S., Patra, R., Surana, S., Subramanian, L., & Brewer, E., Packet loss characterization in Wifi-based Long Distance Networks. *IEEE INFOCOM 2007*, 312-320.
21. Shunra Software Ltd. *The Cloud WAN Emulator*, 2000.
22. Smed, J., Kaukoranta, K., and Hakonen, H. *A Review on Networking and Multiplayer Computer Games*. Technical Report 454, Turku Centre for Computer Science, 2002.
23. Peterson, L., and Davie, B., *Computer Networks*, 5th ed., Amsterdam: Morgan Kaufmann, 2011.
24. Perkins, C., Crowcroft, J., Notes on the use of RTP for shared workspace applications, *ACM SIGCOMM Computer Communication Review*, 30, 2, 35-40.
25. Perkins, C., Hodson, O., Hardman, V., A survey of packet loss recovery techniques for streaming audio, *IEEE Network*, September 1998, 40-48.
26. Quax, P., Monsieurs, P., Lamotte, W., De Vleeschauwer, D., and Degrande, N. Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game. *Proc. NetGames 2004*, 152-156.
27. Salyers, D., Striegel, A., and Poellabauer, C., Wireless reliability: Rethinking 802.11 packet loss, *Proc. IEEE WoWMoM 2008*, 1 – 4.
28. Savery, C, Graham, N., and Gutwin, C., The human factors of consistency maintenance in multiplayer computer games, *Proc. Group 2010*, 187-196.
29. Stuckel, D., and Gutwin, C., The effects of local lag on tightly-coupled interaction in distributed groupware, *Proc. CSCW 2008*, 447-456.
30. Vaghi, I., Greenhalgh, C., Benford, S. Coping with inconsistency due to network delays in collaborative virtual environments. *Proc. VRST 1999*, 42-49.