Experiments in Composing Proxy Audio Services for Mobile Users

Philip K. McKinley, Udiyan I. Padmanabhan, and Nandagopal Ancha

Software Engineering and Network Systems Laboratory
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824 USA
{mckinley,padmana3,anchanan}@cse.msu.edu

Abstract. This paper describes an experimental study in the use of a composable proxy framework to improve the quality of interactive audio streams delivered to mobile hosts. Two forward error correction (FEC) proxylets are developed, one using block erasure codes, and the other using the GSM 06.10 encoding algorithm. Separately, each type of FEC improves the ability of the audio stream to tolerate errors in a wireless LAN environment. When composed in a single proxy, however, they cooperate to correct additional types of burst errors. Results are presented from a performance study conducted on a mobile computing testbed.

1 Introduction

The large-scale deployment of wireless communication services and advances in portable computers are quickly making "anytime, anywhere" computing into a reality. One class of applications that can benefit from this expanding and varied infrastructure is collaborative computing. Examples include computer-supported cooperative work, computer-based instruction, command and control, and mobile operator support in military/industrial installations. A diverse infrastructure enables individuals to collaborate via widely disparate technologies, some using workstations on high-speed local area networks (LANs), and others using wireless handheld/wearable devices.

Collaborative applications differ widely in their quality-of-service requirements and, given their synchronous and interactive nature, they are particularly sensitive to the heterogeneous characteristics of both the computing devices and the network connections used by participants. One approach to accommodating heterogeneity is to introduce a layer of adaptive middleware between applications and underlying transport services [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]. The appropriate middleware framework can help to insulate application components from platform variations and changes in network conditions. Moreover, a properly designed middleware framework can facilitate the development of *new* applications through software reuse and domain-specific extensibility.

We are currently conducting a project called *RAPIDware* that addresses the design and implementation of middleware services for dynamic, heterogeneous

R. Guerraoui (Ed.): Middleware 2001, LNCS 2218, pp. 99–120, 2001.

environments. A major goal of the RAPIDware project is to develop adaptive mechanisms and programming abstractions that enable middleware frameworks to execute in an autonomous manner, instantiating and reconfiguring components dynamically in response to the changing needs of client systems. Moreover, the RAPIDware methodology is intended to apply not only to communication protocols, but also to fault tolerance components, security services, and reconfigurable user interfaces.

For RAPIDware to achieve these goals, it must be possible to compose and reconfigure middleware services at run time. Such adaptability is particularly important in *proxy servers*, which are often used to mitigate the limitations of mobile hosts and their wireless connections [17,18,19,20,9,12,21]. Adopting the terminology of the IETF Task Force on Open Pluggable Edge Services (OPES) [22], we view RAPIDware proxies as composed of many *proxylets*, which are functional components that can be inserted and removed dynamically at run time without disturbing the network state. Moreover, it should not be necessary to compile these components into the proxy code *a priori*, but instead they should be mobile components that are uploaded into proxies at run time.

Earlier in the RAPIDware project, we designed a composable proxy framework based on detachable Java I/O streams [23]. The framework enables proxylets such as filters and transcoders to be dynamically inserted, deleted, and reordered on extant data streams. As such, detachable streams provide the "glue" needed to support the dynamic composition of proxy services. In this paper, we demonstrate how this framework can be used to combine two independent forward error correction (FEC) proxylets, one using block erasure codes [24], and the other using the GSM 06.10 encoding algorithm designed for wireless telephones [25]. Separately, each type of FEC improves the ability of the audio stream to tolerate errors in a wireless LAN environment. However, by simply plugging both proxylets into the framework, they cooperate to correct additional types of burst errors occurring on the wireless network.

The remainder of the paper is organized as follows. In Section 2, we discuss the RAPIDware project and its foundations. Section 3 reviews the design and operation of detachable streams. Section 4 describes the individual operation of the two audio proxylets and their combined functionality when coupled in a single proxy server. Section 5 presents results of an experimental study on a mobile computing testbed. Section 6 discusses related work on composable proxy services. Section 7 presents our conclusions and discusses future directions for the RAPIDware project.

2 Background

The RAPIDware project evolved from our prior work on Pavilion [26], an object-oriented framework to support synchronous web-based collaboration. Like other collaborative frameworks, Pavilion can be used in a default mode, in which it operates as a collaborative web browser [27]. In addition, Pavilion enables a developer to construct new multi-party applications by reusing and extending existing components: browser interfaces, communication protocols, a leadership

protocol for session floor control, and a variety of proxy servers. Pavilion uses proxy servers for several tasks: transcoding and filtering of data streams to reduce bandwidth and load on mobile clients [28], data caching for memory-limited handheld devices [29], forward error correction for real-time isochronous communication [30,31] and reliable data delivery on wireless networks [21].

The RAPIDware project extends Pavilion by developing programming abstractions and supporting mechanisms to *automate* the instantiation and reconfiguration of middleware components, such as proxy services, in order to accommodate resource-limited hosts and dynamic network conditions. A key principle in RAPIDware is to separate adaptive middleware components from non-adaptive, or *core*, middleware services, thereby facilitating dynamic reconfiguration of components at run time. Towards this end, we are designing an extensible set of adaptive components, which we refer to as *raplets*. Figure 1 depicts a simple example of the intended relationship among raplets and other components in a collaborative application. Shown are three types of systems connected by several data streams. Two of the systems use proxy servers to transcode or otherwise modify data prior to its transmission on wireless links.

RAPIDware uses two main types of raplets, *observers* and *responders*, to accommodate heterogeneity and adapt to variations in conditions. The observers collectively monitor the state of the system. When an observer detects a relevant event, the observer either instantiates a new responder or requests an extant responder to take appropriate action to address the event. Example events include changes in the quality of a network connection, disparities among collaborating devices, and changes in user/application preferences or policies. Responder raplets are programmed to handle such events by instantiating new components or modifying the behavior of a communication protocol or user interface.

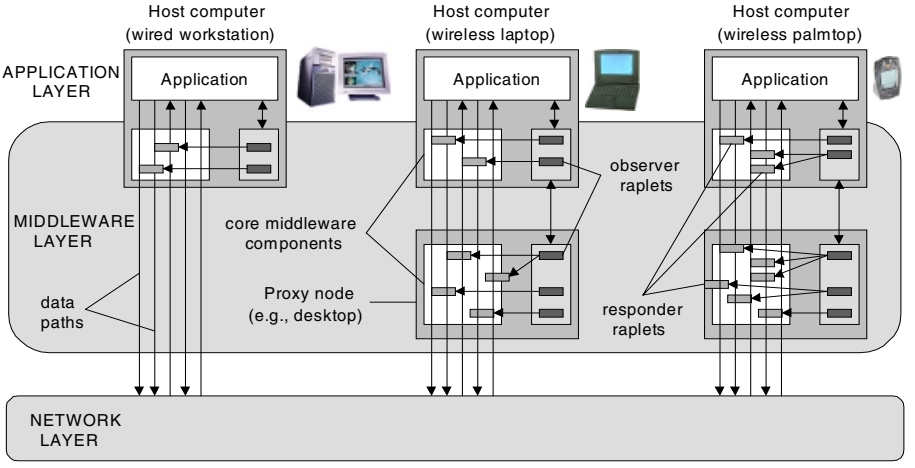


Fig. 1. Configuration of RAPIDware adaptive middleware components.

Besides RAPIDware, several other projects have addressed the issue of configurable and adaptive proxy services for mobile hosts [9,12,32]; these designs enable filters and transcoders to be configured at run time in order to match the capabilities of users devices and networks (see Section 6). If all possible proxylets are known to the proxy *a priori*, then this task is relatively simple. However, it may be difficult to predict the possible variations and sources of proxy services that will be needed. In RAPIDware, we seek to create a framework that enables third-party proxylets to be authenticated and dynamically inserted into an existing proxy. We focus on "lightweight" proxies, typically executed on workstations and other hosts accessible to the mobile user. To manipulate data streams sent to and from clients systems, we require mechanisms that enable the stream to be disconnected and redirected to another piece of code, without compromising the integrity of the data or that of the network state. Next, we review the operation of detachable Java I/O streams, which provide this functionality.

3 Framework Overview

To illustrate the intended operation of RAPIDware proxy components, let us consider the example configuration shown in Figure 2. Suppose that a proxy server is instantiated to support one or more mobile users, connected via a wireless LAN, who are participating in a collaborative session with other users on the wired network. Among other duties, such a proxy might receive live audio/video streams on the wired network, pass them to proxylets that transcode them into a lower bandwidth formats, and forward the new streams on the wireless network. We use the term *filter* to refer to this particular subclass of proxylet, which includes the audio FEC proxylets described in this paper.

Now let us assume that the user wants to maintain the connection as she moves from her office (near the wireless access point) to a conference room down the hall. In such environments, the packet loss characteristics can change dramatically over a distance of only several meters [21]. When losses rise above a given level, the RAPIDware system should insert an FEC filter into such

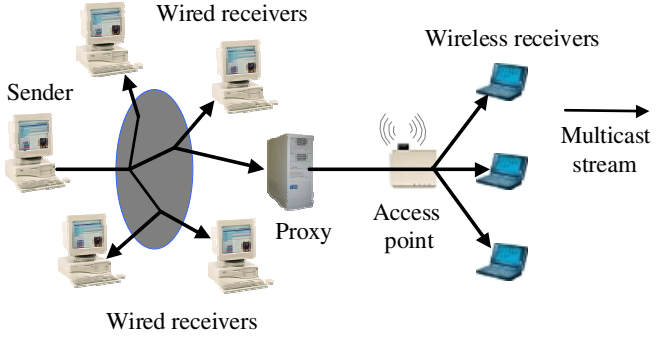


Fig. 2. Proxy configuration for nodes on a wireless LAN.

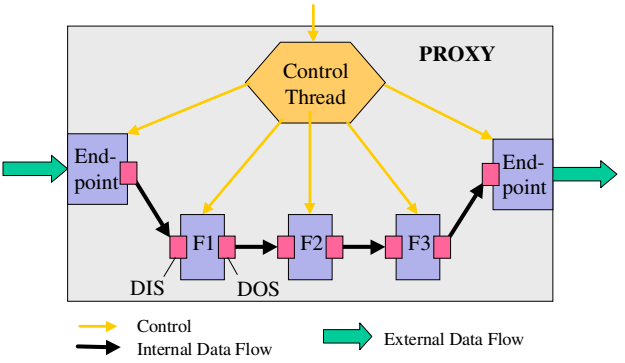


Fig. 3. Composition of proxy filters.

data streams in order to make them stream more resilient to losses. However, the insertion should not disturb the connections to the data sources. Moreover, since the FEC filter may be data-specific (e.g., placing more redundancy in I frames than in B frames in MPEG streams [33]), we need to consider the format of the stream in order to start the FEC filter at the proper point in the stream. Finally, it is possible that the application itself was dynamically downloaded to the mobile host, in which case the associated filter for low-bandwidth connections may not have been known to the proxy in advance. In this case, the filter would need to be fetched from a repository and instantiated on the proxy.

To support such functionality, we began by designing objects and interfaces to enable filters to be inserted, deleted, and chained together [23]. Figure 3 depicts the resulting software structure of a RAPIDware proxy and its operation on a single data stream. The proxy receives and transmits the stream on *EndPoint* objects, which encapsulate the actual network connections. Each EndPoint has an associated thread that reads or writes data on the network, depending on the configuration of the EndPoint. A ControlThread object is responsible for managing the insertion, deletion, and ordering of the filters associated with the stream. In this example, the proxy comprises three filters, F1, F2, and F3. The key support mechanisms are detachable stream objects, namely, DetachableInputStream (DIS) and DetachableOutputStream (DOS). The DIS and DOS are used for all communication among filters, and between filters and EndPoints. The DIS and DOS can be stopped (paused), disconnected, and reconnected, enabling the dynamic redirection and modification of data streams. Now, let us briefly describe the main classes that make up the framework.

DetachableOutputStream and DetachableInputStream. These classes are based on the the Java PipedOutputStream and PipedInputStream classes, respectively. DetachableOutputStream extends the base java.io.OutputStream class, and DetachableInputStream extends the base java.io.InputStream class. In addition to overriding many of the base class methods, we have also included additional state variables and methods to implement the functionality needed to support com-

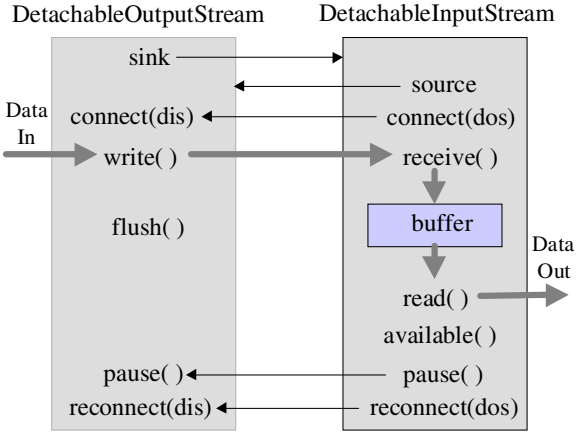


Fig. 4. Configuration of detachable streams.

posable filters. Figure 4 illustrates the relationship between a DetachableOutputStream (DOS) and a DetachableInputStream (DIS). The connect() method is used to associate a specific output stream with a specific input stream. Among other initializations, the connect() method sets DOS.sink and DIS.source variables so as to identify the other half of the connection. The DIS.connect() method simply calls the DOS.connect() method, which actually does this initialization.

As with their piped counterparts, the data written to the stream is buffered at the DIS side. An invocation of the DOS.write() method results in a call to the DIS.receive() method, which places the data in the buffer. The DIS.available() method returns the number of bytes currently in the buffer, and data is retrieved from the buffer using the DIS.read() method. The DOS.flush() method can be used to force any buffered output bytes to be written out, and notifies any readers that bytes are waiting in the pipe. Unlike the PipedOutputStream and PipedInputStream classes, both the DOS and DIS can be temporarily paused and reconnected to other streams. The pause() method has to be called before any actual disconnection and switching of the data stream. The method blocks attempts to write to the buffer and ensures that all the data has been read from the buffer. It also sets flags indicating that the two sides are no longer connected. Once the pause method returns, the reconnect() method can be used to attach the DIS and DOS to other filters. If the buffer has not yet emptied, the caller is suspended until the buffer becomes empty. The reconnect() method checks whether the call is valid (not still in the connected state) and then mimics the actions of the connect() method in setting several global variables.

ControlThread. This class is used to manage the configuration of filters on a given stream supported by the proxy. The class maintains the Filter Vector, a dynamic array that holds references to the currently configured filters. The class implements methods to insert and delete filters from the Filter Vector, as well as methods that allow the ControlManger class to query about the available filters

and the methods they support. The ControlThread receives commands from across the network, either from the mobile client, from an application server, or from the control manager.

Filter and FilterContainer. The base class for proxylets is Detachable.Proxylet. Any proxylet that is to be used in the framework needs to extend this base class. The Proxylet class extends the Thread class and thus is inherently runnable. The Filter class extends the base class and is meant to be further extended by all proxy filters that are to be run in the proposed framework; see Figure 5. The author of a filter would write the functional code as the run() method of the filter. The Filter class contains a DIS and a DOS object, along with their corresponding standard references, called *DIS* and *DOS*. The ControlThread uses these references to manipulate the stream connections. A group of methods (e.g., setDIS, setDOS, getid) is used to establish references to the DIS and DOS in the filter code itself. The FilterContainer class is used to hold an array of Filter objects. This functionality is required when new filters are uploaded into the framework from remote hosts. The FilterContainer class has methods to obtain the number of Filters available and an enumeration method to return a String enumeration of the Filter objects names.

EndPoint. These are extensions of Filters that are instantiated by the ControlThread for providing input and output services to the framework. If the I/O is network-based, then the EndPoint objects would be a EndPointSocketReader and EndPointSocketWriter. If the I/O is a non-network stream then we would

```
public class Filter extends Proxylet implements Serializable, Cloneable {
    String idString = new String("NA/");  // the filter identifier

    // The DIS, which will be manipulated by the ControlThread
    public final DetachableInputStream DIS = new DetachableInputStream();

    // The DOS, which will be manipulated by the ControlThread
    public final DetachableInputStream DOS = new DetachableOutputStream();

    private int objid = -1;  // An object id for convenience

    // The setDIS and setDOS methods allow on to set up their own
    // references names to the actual DIS and DOS
    public DetachableInputStream setDIS(){
        return(this.DIS);    }

    public DetachableOutputStream setDOS(){
        return(this.DOS);    }
```

Fig. 5. Excerpt from the Filter class.

use an EndPointStreamReader and EndPointStreamWriter. Each EndPoint contains an active thread that handles I/O to and from the proxy. Combined with the ControlThread, two EndPoints comprise a "null" proxy, that is, one that simply forwards data without modifying it. Upon insertion of a filter between the EndPoints, the stream is redirected through the new code.

ControlManager. As described earlier, observer and responder components provide adaptive functionality in RAPIDware, based on predefined user preferences and device/network descriptors. To test the behavior of RAPIDware filters and related components, however, we found it useful to develop a user interface that can be used to manage RAPIDware-based collaborative sessions. The ControlManager class has a Swing-based GUI designed for this purpose. Based on responses to queries, the ControlManager constructs a graphical representation of the state of the proxy, including the current configuration of filters, based on the methods available in the ControlThread. This design enables the same ControlManager to be used with different types of proxies. The current ControlManager GUI is shown in Figure 6. Although it is rather primitive and used only for our testing, we intend to improve it as the RAPIDware project progresses. The interface displays the current configuration of proxy filters and has pull-down menus and dialog text boxes that allow an administrator to insert and remove filters at specified locations in a given stream. The ControlManager uses serialization to deliver new filters to the proxy, as they are requested.

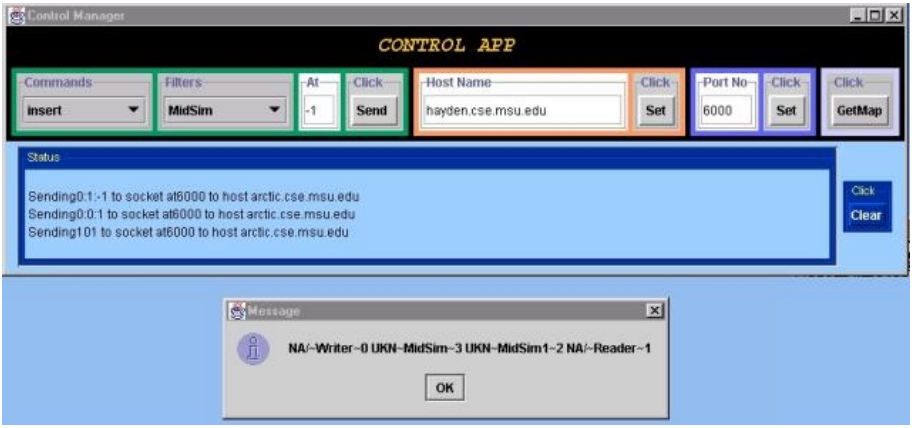


Fig. 6. Screen capture of ControlManager.

4 The Audio Filters and Their Composition

To evaluate the operation and performance of the RAPIDware DIS/DOS proxy framework, as well to determine which new features are needed, we are constructing several filters and other proxylets for use in the framework. For example, we are porting proxy services from Pavilion (such as video transcoding and reliable

muliticasting services) to the new framework, and we are developing new proxylets related to security management and handoff of applications among different graphical displays. As part of this testing, we have developed two different audio FEC filters, each of which can be independently inserted in a running audio stream. Moreover, we note that chaining together the two filters can provide a level of error correction beyond what either filter can provide separately. We begin by discussing the packet loss characteristics of wireless LANs (WLANs) and their effects on interactive audio streaming, followed by details of the two audio FEC filters and a description of their combined operation.

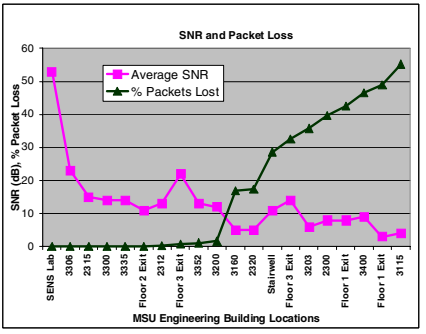
Characteristics of Wireless LANs. The performance of group communication services, such as audio multicasting for collaborative applications, is affected by four main characteristics of WLANs. First, the packet loss rates are highly dynamic and location-dependent [21]. Figure 7(a) demonstrates this behavior by plotting the relationship between signal-to-noise ratio (SNR) and packet loss rate during a short excursion within range of the wireless access point in our laboratory. The results demonstrate the highly variable loss rate that can occur in such environments, and the SNR values quickly drop below the level of 20 dB that is typically considered acceptable.

Second, the loss characteristics of a WLAN are very different from those of a wired network. In a wired domain, losses occur mainly due to congestion and subsequent buffer overflow. In wireless domain, however, losses are more commonly due to external factors like interference, alignment of antennae, ambient temperature, and so on. Figures 7(b) and (c) show example burst error distributions for two locations near our laboratory, where our wireless access point is located. Location 1 is just outside our laboratory, and location 2 is approximately 25 meters down a corridor. In both cases, while some large bursts occur, many are very short, and most "burst" errors comprise a single packet loss. To minimize the loss rate in terms of bytes, smaller packets are preferred, as shown in Figure 7(d). Apparently, the errors within larger packets are relatively localized, so that by sending several smaller packets, some number of them will be received successfully, whereas the larger packet would be lost.

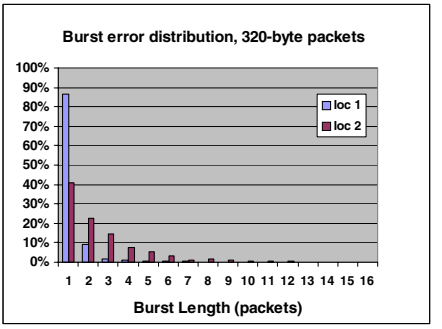
Third, the 802.11b CSMA/CA MAC layer provides RTS/CTS signaling and link-level acknowledgements for unicast frames, but not for multicast frames. The result is a higher packet loss rate as observed by applications using UDP/IP multicast, as opposed to UDP unicast. Figure 8 demonstrates this behavior for a typical location just outside our laboratory. As a result, multicast transmissions require more redundancy from the application level.

Fourth, since the wireless channel is a shared broadcast medium, it is important to minimize the amount of feedback from receivers. Simultaneous responses from multiple receivers can cause channel congestion and burden the access point, thereby hindering the forward transmission of data. Thus, it is desirable to use proxy-based FEC instead of proxy-based retransmissions for real-time communication such as interactive audio streams.

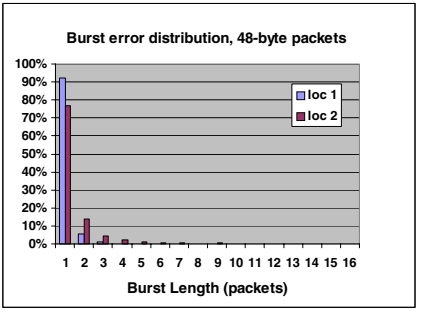
Audio Filter Using Block-Oriented FEC. Our first audio filter uses an FEC mechanism that can be applied to any data type. It recovers packets that have



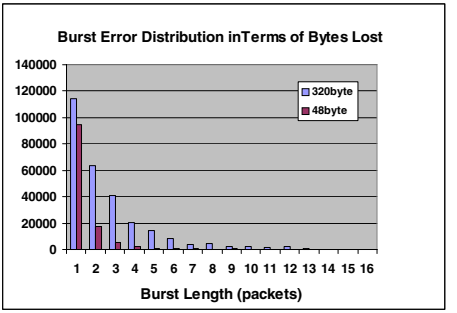
(a) packet loss vs. SNR



(b) burst distr., 320-byte packets



(c) burst distr., 320-byte packets

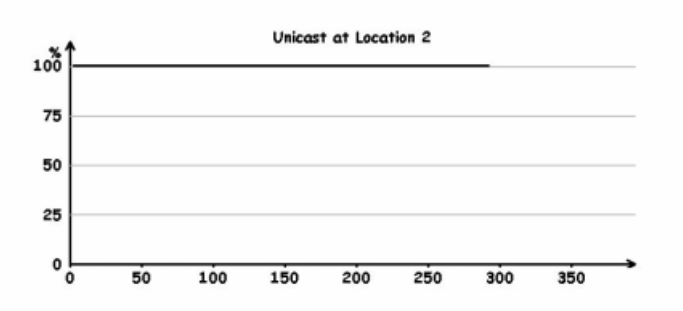


(d) 320-byte vs. 48-byte packets

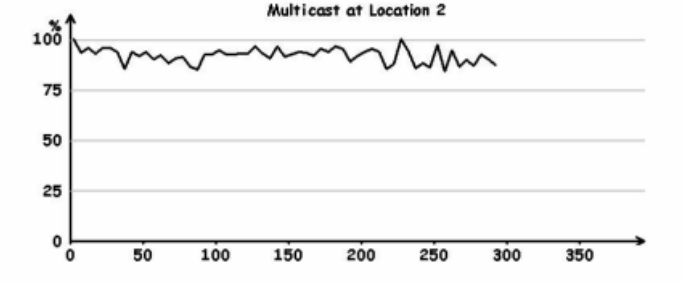
Fig. 7. Typical characteristics of a WLAN channel.

been "erased" due to an error detected by the CRC check in the data link layer. As shown in Figure 9, an (n, k) *block erasure code* converts k source packets into n encoded packets, such that any k of the n encoded packets can be used to reconstruct the k source packets [34]. In this paper, we use only *systematic* codes, which means that the first k of the n encoded packets are identical to the k source packets. We refer to the first k packets as *data* packets, and the remaining $n-k$ packets as *parity* packets. Each set of n encoded packets is referred to as a *group*. The advantage of using block erasure codes for multicasting is that a single parity packet can be used to correct independent single-packet losses among different receivers [24]. These codes are lossless, in that a successful decoding produces exactly the original data.

Recently, Rizzo [24] studied the feasibility of software encoding/decoding for packet-level FEC, using a particular block erasure code called the Depending on the values of k and $n-k$, Rizzo showed that this code can be efficiently executed on many common microprocessors. Rizzo's public domain FEC encoder/decoder library [24] is implemented in C and has been used in many projects involving



(a) UDP unicast reception rate



(b) UDP/IP multicast reception rate

Fig. 8. Sample packet loss traces for UDP/IP unicast and multicast.

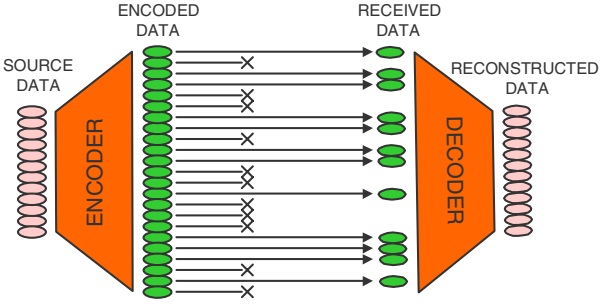


Fig. 9. Operation of FEC based on block erasure codes.

multicast communication [35,36,37,38,39], including our own prior studies [21, 30,31]. Given the emphasis on portability and code mobility in the RAPIDware project, however, we decided to switch to an open-source Java implementation of Rizzo's FEC library, available from Swarmcast [40]. In general, we found the Swarmcast library to provide a convenient interface and good performance. Although considerably slower than the C implementation, the Java version was able to satisfy real-time audio encoding and decoding requirements on systems with modest processing power.

Figure 10 shows the operational schematic of the major components of the audio application when this FEC filter is running. The audio recorder was built as a pure Java application making use of the Java Sound API. Specifically, the recording thread uses the javax.sound.* classes to read audio data from a workstation's sound card and send it to the proxy via the wired network. The encoding of the data is 16 bits per sample, PCM signed, at the standard rate of 8000 Hz over a mono channel. The audio receiver uses one thread to read data from the network and store it in a circular buffer. A second thread reads the data and uses the Java Sound API to play it.

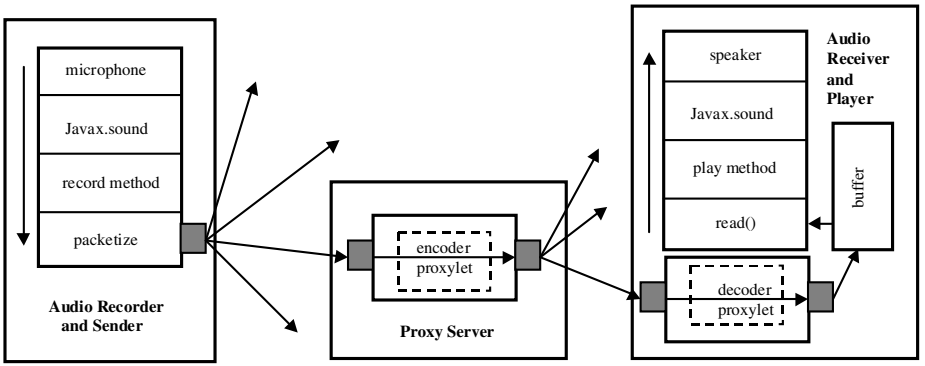
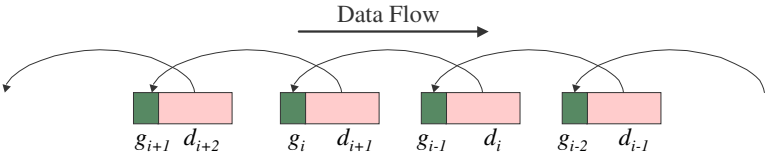


Fig. 10. Operational block diagram of the streaming audio configuration.

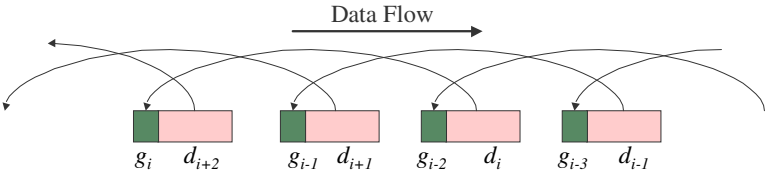
For data streams directed toward the client system, the encoder is instantiated on the proxy and the decoder on the client (the placement is reversed for outbound streams). The encoder and decoder filters have the same basic construction and simply invoke different methods in the Swarmcast FEC library. The encoder filter creates the FEC encoder by instantiating a class of the FECCodefactory's default FEC codec. The thread then collects k packets and constructs references to these. The FEC encoder is then called, which returns n packets contained in a new reference buffer. These packets are labeled with a group identifier and are written to the data stream. The decoder at the client requires any k packets in a given group in order to decode the original k data packets. The decoder thread thus reads up to k packets in a given group, after which additional packets are discarded. This data is passed to the decode

method of the FEC codec, which returns the k original data packets, which are forwarded to the client application.

GSM Audio Filter. While block-oriented FEC approaches are effective in improving the quality of interactive audio streams on wireless networks [30], the group sizes must be relatively small in order to reduce playback delays. Hence, the overhead in terms of parity packets is relatively high. An alternative approach with lower delay and lower overhead is *signal processing based FEC (SFEC)* [41, 42], in which a lossy, compressed encoding of each packet i is piggybacked onto one or more subsequent packets. If packet i is lost, but one of the encodings of packet i arrives at the receiver, then at least a lower quality version of the packet can be played to the listener. The parameter θ is the offset between the original packet and its compressed version. Figure 11 shows two different examples, one with $\theta = 1$ and the other with $\theta = 2$. As mentioned, it is also possible to place multiple encodings of the same packet in the subsequent stream, for example, using both $\theta = 1$ and $\theta = 3$.



(a) GSM encoding with $\theta = 1$



(b) GSM encoding with $\theta = 2$

Fig. 11. Different ways of using GSM encoding on a packet stream.

The SFEC RAPIDware filter that we developed uses GSM 06.10 encoding [25] for generating the redundant copies of packets. The full rate speech codec in GSM is described as Regular Pulse Excitation with Long Term Prediction (GSM 06.10 RPE-LTP). Although GSM is a CPU-intensive coding algorithm – it is 1200 times more costly than normal PCM encoding [42] – the bandwidth overhead is very small. Specifically, the GSM encoding creates only 33 bytes for a PCM-encoded packet containing up to 320 bytes (160 samples).

Our filter uses a freeware Java version of the GSM codec available from Tritonus (www.tritonus.org).

The filter can work in one of two ways. The first is to piggyback encoded data on subsequent packets, as shown in Figure 11. The second is to compute encodings for multiple packets and create a new packet of encoded data, to be inserted in the stream at a later point. This second method is useful when it is important to keep packet sizes small, as in a wireless LAN.

Combining the Audio Filters. Either filter can be inserted separately into an audio stream. However, we can also insert both filters, as shown in Figure 12 (in the figure and in the remainder of the paper, we'll refer to the filters simply as "GSM" and "FEC"). If the direction of the audio channel were reversed, then the encoders would reside on the client, and the decoders on the proxy.

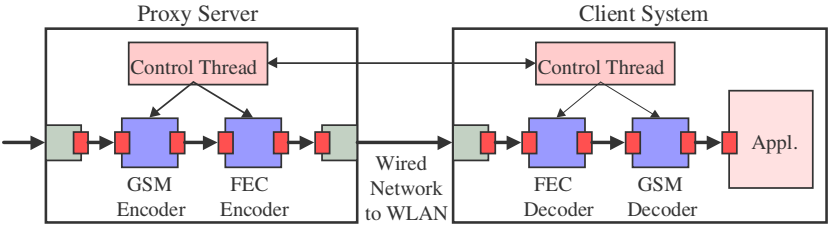


Fig. 12. Configuration of audio filters on proxy and client.

Figure 13 shows a particular example of the two encoders working together. The GSM encoder is configured to operate on a packet basis, computing a GSM encoding for every three data packets and inserting the new packet after $3 \times \theta$ data packets in the following packet stream. Each group of four packets (three data packets and one GSM packet) is forwarded to the FEC filter, which is configured to compute two parity packets using a (6,4) block erasure code. The (6,4) code can recover at most two lost packets per group, hence covering the most common packet loss cases, and does so without any loss in quality. Combining FEC and GSM code can tolerate relatively long isolated burst errors of up to $(\theta + 2)n - 2k$ packets, depending on the location of the lost packets relative to group boundaries. Of course, if GSM instead of FEC is used to reconstruct a packet, the quality of the resulting signal will be lower than that of the original.

5 Experimental Evaluation

In order to study the operation and performance of audio filters separately and in combination, we conducted a set of experiments on the mobile computing testbed in our Software Engineering and Network Systems (SENS) Laboratory. Results of using block erasure codes alone are described elsewhere [23,30], so here we concentrate on the GSM encoder and the combination of the two methods. Detailed evaluations are described in a companion technical report [43].

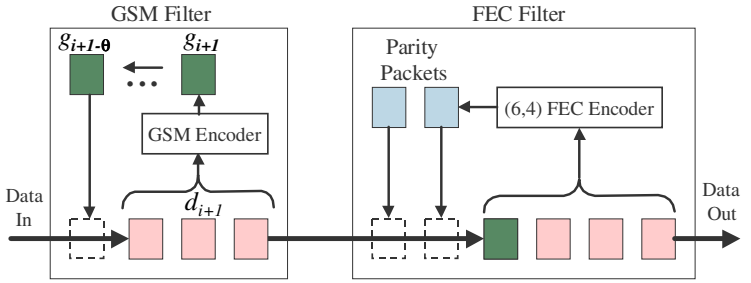


Fig. 13. Operation of combined audio filters.

Testing Environment. The mobile testbed currently includes conventional workstations connected by a 100 Mbps Fast Ethernet switch, three 802.11 WLANs (Lucent WaveLAN, Proxim RangeLAN2, and Cisco/Aironet), and several mobile handheld and laptop computer systems. All tests reported here were conducted on the Aironet WLAN, which uses direct sequence spread spectrum signaling and has a raw bit rate of 11 Mbps. We used both wired desktop PCs and wireless laptop PCs as participating stations in the experimental configuration depicted in Figure 14. The sender, proxy, and control manager were executed on dual-processor 400/450 MHz desktop workstations, while the mobile nodes were 300 MHz laptops equipped with Aironet network interface cards. The Aironet access point and the participating wired stations were located in our laboratory, while the locations of the mobile nodes were varied. Although the proxy multicasts the audio stream on the WLAN, here we report the results for a single receiver.

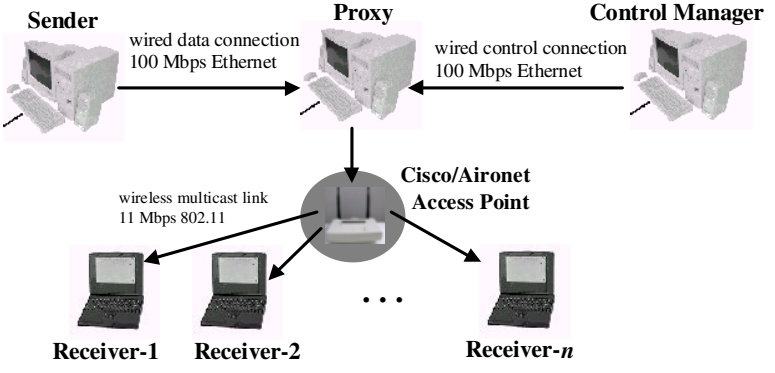
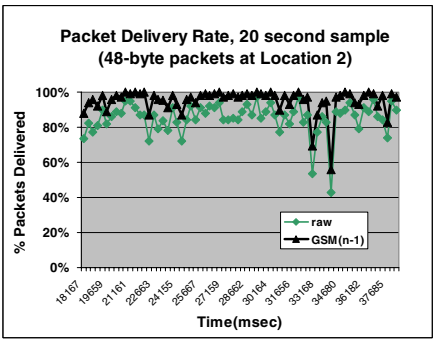


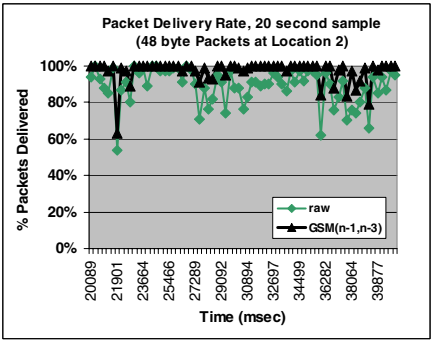
Fig. 14. Physical configuration of experimental components.

Initially, both the proxy and client are configured as "null" filters, as the Endpoints simply read and retransmit data. In an actual RAPIDware environment, observer threads at the client would monitor the packet loss rate and burst length distribution, and would instruct the ControlThread on the proxy when to insert the FEC or GSM filter. In order to control the testing, however, we used the Control Manager GUI to insert the filters manually.

GSM Experiments. We tested the GSM filter in isolation, setting θ to different values and using both single and double copies of the encoded data. In all cases, small 48-byte packets produced considerably better results than 320-byte packets, so we report only the former here; many additional results can be found in [43]. Figure 15 shows a sample of the results. In Figure 15(a), we placed a single encoded copy of each packet i in its successor packet $i+1$. In Figure 15(b), we placed one encoded copy in packet $i+1$ and one in packet $i+3$. Using multiple copies produces a clear advantage in terms of packet delivery rate.



(a) $\theta = 1$, 48-byte packets

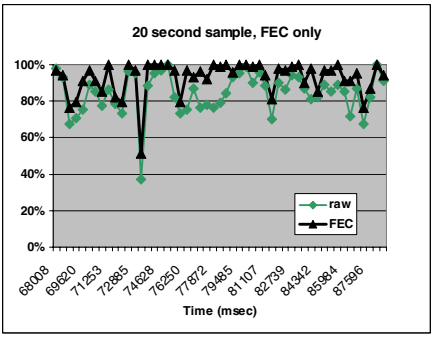


(b) $\theta = 1, 3$, 48-byte packets

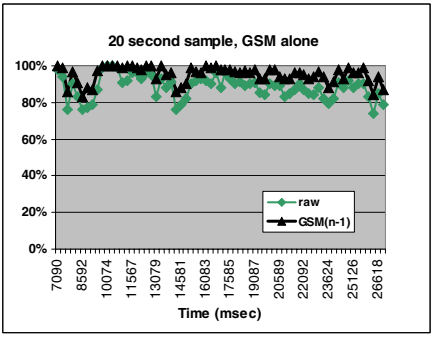
Fig. 15. Sample traces results for the GSM filter with 48-byte packets.

Figures 16(a) through (c), respectively, show sample traces of using the FEC(8,4) and GSM(n-1) filters, alone and in combination. The combination appears to be most effective in recovering data, and this result is confirmed by Figure 16(d), where we averaged the results of 5 two-minute runs and computed the packet delivery rate for each of the methods at a particular location in our building. By combining the two filters, we are able to reconstruct 97% of the audio data, even though the raw packet delivery rate at this location was only 83%.

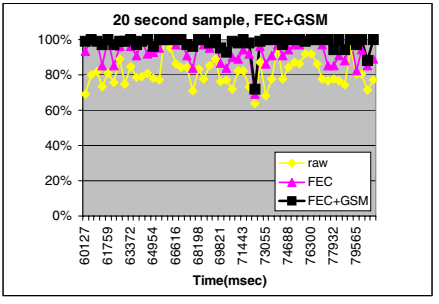
Finally, we conducted a set of experiments near the periphery of the wireless cell (Location 3), where large burst errors are more frequent. In these tests, we again combined the FEC filter with the GSM filter, but we configured the latter to use two values of θ, both 1 and 3, which enables it to correct longer burst



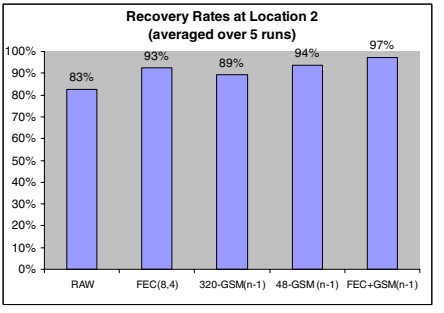
(a) FEC(8,4) alone



(b) GSM(n-1) alone



(c) FEC(8,4)+GSM(n-1)



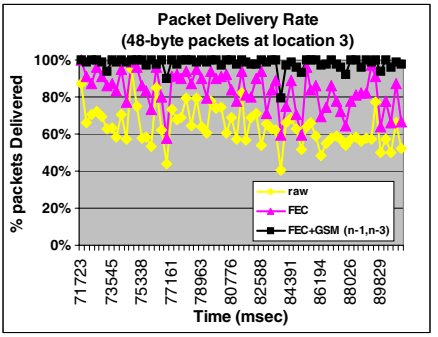
(d) performance comparison

Fig. 16. Effects of FEC, GSM filters and their composition.

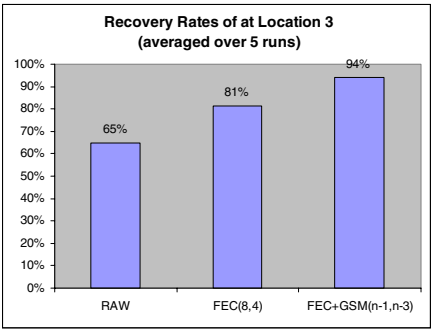
errors. Figure 17(a) shows a short sample trace. Despite the fact that the raw delivery rate sometimes falls below 50%, the combination of filters is extremely effective in recovering the lost data. Figure 17(b) shows the average results of 5 two-minute runs. At this location the raw delivery rate was only 65%, FEC alone raised the rate to 81%, and the GSM filter further improved the rate to 94%. The 13% GSM improvement over FEC alone compares with only a 4% improvement at Location 2. We conclude that, while both types of filters improve the quality of the audio channel, near the cell periphery, they are almost equally important. These results demonstrate the utility of being able to compose two different proxylets easily and dynamically within a single proxy framework.

6 Related Work

In recent years, numerous research groups have addressed the issue of adaptive middleware frameworks that can accommodate dynamic, heterogeneous infras-



(a) sample trace



(b) performance comparison

Fig. 17. Effects of FEC and GSM filters near the wireless cell periphery.

tructures. Examples include CEA [1], MOOSCo [2], BRAIN [3], Squirrel [4], Adapt [5], MOST [6], AspectIX [44], Limbo [45], MASH [10], TAO [11], MobiWare [12], MCF [13], QuO [14], MPA [9], Odyssey [15], and DaCapo++ [16] Rover [7], BARWAN [32], and Sync [46]. These projects have greatly improved the understanding of how middleware can accommodate device heterogeneity and dynamic network conditions, particularly in the area of adaptive communication protocols and services. Indeed, several of these projects address dynamic configuration of proxy and/or stream functionality. In the remainder of this section, we discuss four such projects and their relationship to RAPIDware.

In the MobiWare project [12], "mobile filters" can be dispatched to various nodes in the network, or to hosts, in order to achieve bandwidth conservation. Apparently, these filters are established only during handoff from one network to another. The detachable streams discussed herein could be used to extend this functionality so that filters could be reorganized at any time.

In the Adapt project at Lancaster [5], CORBA is extended to support open bindings, which enable manipulation and reconfiguration of communication paths through the use of object graphs. This powerful mechanism could be used directly to implement dynamically composable proxy services. In contrast to a CORBA-based design, we sought in this subproject to determine the minimal level of functionality needed to provide dynamic composition of communication stream components. We offer detachable Java I/O streams as a possible solution.

The Berkeley TranSend proxy is based on the TACC model [8], in which *workers* are the active components of the proxy. The TACC server enables workers to be chained together in a manner similar to Unix pipes. Details of the implementation are not available. However, the project focuses on proxies built atop highly available parallel workstation clusters, whereas RAPIDware proxies are intended to be lightweight, on-demand proxies established dynamically on one or more idle workstations available to the user.

The Stanford Mobile People Architecture (MPA) [9] is designed to support person-to-person reachability through the use of *personal proxies*. A key component of the personal proxy is the use of *conversion drivers*, which are configured dynamically to match the capabilities of the user's device and network. The RAPIDware project seeks to develop a general framework for the design of such software. In this case, the detachable stream mechanism and filter container class may provide a useful mechanism for composing such drivers and facilitating their dynamic loading and unloading from across the network.

Finally, we emphasize that this paper has described only a small part of the RAPIDware project. A given external event, such as a sudden decrease in quality on a wireless link, can affect not only communication protocols, but also middleware components associated with fault tolerance, security, and user interfaces. The overall goal of the RAPIDware project is to develop an integrated methodology for middleware adaptability that encompasses not only communication services, but also security policies, fault tolerance actions, and user interface reconfiguration. We will report developments in these areas in future papers.

7 Conclusions and Future Work

In this paper, we have described the use of a detachable Java I/O stream framework to support composition of proxy services. We reviewed the design of the framework, which enables proxylets to be dynamically inserted, deleted, and reordered. Addition and removal of proxy code is achieved without disturbing existing network connections. We demonstrated the use of the framework to support the instantiation and composition of two different audio FEC filters, one using block erasure codes and the other using the GSM 06.10 encoder, and we evaluated their performance on a WLAN testbed. The main contribution of this work is to show that independent proxylets can be composed and can cooperate synergistically, given the proper supporting proxy framework.

Our continuing work in this area addresses several issues: porting of additional proxies to the RAPIDware framework; development of a rules engine to characterize the "composability" of proxylets; and application of RAPIDware concepts to intrusion detection, fault tolerance, and user interfaces handoff. Given the increasing presence of wireless networks in homes and businesses, we envision application of the proposed techniques to improve performance of collaborative applications involving users who roam within a wireless environment.

Further Information. A number of related papers and technical reports of the Software Engineering and Network Systems Laboratory can be found at the following URL: `http://www.cse.msu.edu/sens`.

Acknowledgements. The authors would like to thank Arun Mani, Suraj Gaurav, Peng Ge, and Chiping Tang for their contributions to this work. This work was supported in part by the U.S. Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744. This work was also supported in part by U.S. National Science Foundation grants CDA-9617310, NCR-9706285, CCR-9912407, and EIA-0000433.

References

1. Bacon, J., Moody, K., Bates, J., Hayton, R., Ma, C., McNeil, A., Seidel, O., Spiteri, M.: Generic support for distributed applications. IEEE Computer **33** (2000) 68–76
2. Miranda, H., Antunes, M., Rodrigues, L., Silva, A.R.: Group communication support for dependable multi-user object-oriented environments. In: SRDS Workshop on Dependable System Middleware and Group Communication (DSMGC 2000), Nürnberg, Germany (2000)
3. Burness, L., Kassler, A., Khengar, P., Kovacs, E., Mandato, D., Manner, J., Neureiter, G., Robles, T., Velayos, H.: The BRAIN quality of service architecture for adaptable services. In: Proceedings of the PIMRC 2000, London (2000)
4. Kramp, T., Koster, R.: A service-centered approach to QoS-supporting middleware (Work-in-Progress Paper). In: IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), The Lake District, England (1998)
5. Fitzpatrick, T., Blair, G., Coulson, G., Davies, N., Robin, P.: A software architecture for adaptive distributed multimedia applications. IEE Proceedings - Software **145** (1998) 163–171
6. Friday, A., Davies, N., Blair, G., Cheverst, K.: Developing adaptive applications: The MOST experience. Journal of Integrated Computer-Aided Engineering **6** (1999) 143–157
7. Joseph, A.D., Tauber, J.A., Kaashoek, M.F.: Mobile computing with the Rover toolkit. IEEE Transactions on Computers: Special issue on Mobile Computing **46** (1997)
8. Fox, A., Gribble, S.D., Chawathe, Y., Brewer, E.A.: Adapting to network and client variation using active proxies: Lessons and perspectives. IEEE Personal Communications (1998)
9. Roussopoulos, M., Maniatis, P., Swierk, E., Lai, K., Appenzeller, G., Baker, M.: Person-level routing in the mobile people architecture. In: Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems, Boulder, Colorado (1999)
10. McCanne, S., Brewer, E., Katz, R., Rowe, L., Amir, E., Chawathe, Y., Coopersmith, A., Mayer-Patel, K., Raman, S., Schuett, A., Simpson, D., Swan, A., Tung, T., Wu, D., Smith, B.: Toward a common infrastructure for multimedia-networking middleware. In: Proc. 7th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97), St. Louis, Missouri. (1997)
11. Kuhns, F., O'Ryan, C., Schmidt, D.C., Othman, O., Parsons, J.: The design and performance of a pluggable protocols framework for object request broker middleware. In: Proceedings of the IFIP Sixth International Workshop on Protocols For High-Speed Networks (PfHSN '99), Salem, Massachusetts (1998)
12. Angin, O., Campbell, A.T., Kounavis, M.E., R.R.-F.M. Liao: The Mobiware toolkit: Programmable support for adaptive mobile networking. IEEE Personal Communications Magazine, Special Issue on Adapting to Network and Client Variability (1998)
13. Li, B., Nahrstedt, K.: A control-based middleware framework for quality of service adaptations. IEEE Journal of Selected Areas in Communications **17** (1999)
14. Vanegas, R., Zinky, J.A., Loyall, J.P., Karr, D.A., Schantz, R.E., Bakken, D.E.: QuO's runtime support for quality of service in distributed objects. In: Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), The Lake District, England (1998)

15. Noble, B.D., Satyanarayanan, M.: Experience with adaptive mobile applications in Odyssey. Mobile Networks and Applications **4** (1999) 245–254
16. Stiller, B., Class, C., Waldvogel, M., Caronni, G., Bauer, D.: A flexible middleware for multimedia communication: Design implementation, and experience. IEEE Journal of Selected Areas in Communications **17** (1999) 1580–1598
17. Badrinath, B.R., Bakre, A., Marantz, R., Imielinski, T.: Handling mobile hosts: A case for indirect interaction. In: Proc. Fourth Workshop on Workstation Operating Systems, Rosario, Washington, IEEE (1993)
18. Zenel, B., Duchamp, D.: Intelligent communication filtering for limited bandwidth environments. In: Proc. Fifth Workshop on Hot Topics in Operating Systems, Rosario, Washington (1995)
19. Chen, L., Suda, T.: Designing mobile computing systems using distributed objects. IEEE Communications Magazine **35** (1997)
20. Chawathe, Y., Fink, S., McCanne, S., Brewer, E.: A proxy architecture for reliable multicast in heterogeneous environments. In: Proceedings of ACM Multimedia '98, Bristol, UK (1998)
21. McKinley, P.K., Mani, A.P.: An experimental study of adaptive forward error correction for wireless collaborative computing. In: Proceedings of the IEEE 2001 Symposium on Applications and the Internet (SAINT-01), San Diego-Mission Valley, California (2001)
22. Yang, L., Hofmann, M.: OPES architecture for rule processing and service execution. Internet Draft draft-yang-opes-rule-processing-service-execution-00.txt (2001)
23. McKinley, P.K., Padmanabhan, U.I.: Design of composable proxy filters for mobile computing. In: Proceedings of the Second International Workshop on Wireless Networks and Mobile Computing, Phoenix, Arizona (2001)
24. Rizzo, L.: Effective erasure codes for reliable computer communication protocols. ACM Computer Communication Review (1997)
25. Degener, J., Bormann, C.: The gsm 06.10 lossy speech compression library and its applications (2000) available at `http://kbs.cs.tu-berlin.de/~jutta/toast.html`.
26. McKinley, P.K., Malenfant, A.M., Arango, J.M.: Pavilion: A distributed middleware framework for collaborative web-based applications. In: Proceedings of the ACM SIGGROUP Conference on Supporting Group Work. (1999) 179–188
27. McKinley, P.K., Barrios, R.R., Malenfant, A.M.: Design and performance evaluation of a Java-based multicast browser tool. In: Proceedings of the 19th International Conference on Distributed Computing Systems, Austin, Texas (1999) 314–322
28. Arango, J., McKinley, P.K.: VGuide: Design and performance evaluation of a synchronous collaborative virtual reality application. In: Proceedings of the IEEE International Conference on Multimedia and Expo, New York (2000)
29. McKinley, P.K., Li, J.: Pocket Pavilion: Synchronous collaborative browsing for wireless handheld computers. In: Proceedings of the IEEE International Conference on Multimedia and Expo, New York (2000)
30. McKinley, P.K., Gaurav, S.: Experimental evaluation of forward error correction on multicast audio streams in wireless LANs. In: Proceedings of ACM Multimedia 2000, Los Angeles, California (2000) 416–418
31. Ge, P., McKinley, P.K.: Experimental evaluation of error control for video multicast over wireless LANs. In: Proceedings of the Third International Workshop on Multimedia Network Systems, Phoenix, Arizona (2001)

32. Katz, R. H., Brewer, E. A., et al.: The Bay Area Research Wireless Access Network (BARWAN). In: Proceedings Spring COMPCON Conference. (1996)
33. Xu, D., Li, B., Nahrstedt, K.: Qos-directed error control of video multicast in wireless networks. In: Proceedings of IEEE International Conference on Computer Communications and Networks. (1999)
34. McAuley, A.J.: Reliable broadband communications using burst erasure correcting code. In: Proceedings of ACM SIGCOMM. (1990) 287–306
35. Rizzo, L., Vicisano, L.: RMDP: An FEC-based reliable multicast protocol for wireless environments. ACM Mobile Computer and Communication Review **2** (1998)
36. Nonnenmacher, J., Biersack, E.W., Towsley, D.: Parity-based loss recovery for reliable multicast transmission. IEEE/ACM Transactions on Networking **6** (1998) 349–361
37. Huitema, C.: The case for packet level FEC. In: Proceedings of IFIP 5th International Workshop on Protocols for High-Speed Networks (PfHSN'96). (1996) 110–120 INRIA, Sophia Antipolis, France.
38. Gemmell, J., Schooler, E., Kermode, R.: A scalable multicast architecture for one-to-many telepresentations. In: Proceedings of IEEE International Conference on Multimedia Computing Systems. (1998) 128–139
39. Kermode, R.: Scoped Hybrid Automatic Repeat ReQuest with Forward Error Correction (SHARQFEC). In: Proceedings of ACM SIGCOMM. (1998) Vancouver, Canada.
40. Swarmcast: Release notes for Java FEC v0.5. `http://www.swarmcast.com` (2001)
41. Podolsky, M., Romer, C., McCanne, S.: Simulation of FEC-based error control for packet audio on the Internet. In: Proceedings of IEEE INFOCOM'96, San Francisco, California (1998)
42. Bolot, J.C., Vega-Garcia, A.: Control mechanisms for packet audio in Internet. In: Proceedings of IEEE INFOCOM'96, San Francisco, California (1996) 232–239
43. McKinley, P.K., Padmanabhan, U., Ancha, N.: Performance evaluation of audio FEC on wireless LANs. Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, in preparation (2001)
44. Hauck, F., Becker, U., Geier, M., Meier, E., Rastofer, U., Steckermeier, M.: AspectIX: A middleware for aspect-oriented programming. Technical Report TR-I4-98-06, Computer Science Department, Friedrich-Alexander-University, Erlangen-Nürnberg, Germany (1998)
45. Blair, G.S., Davies, N., Friday, A., Wade, S.P.: Quality of service support in a mobile environment: An approach based on tuple spaces. In: Proceedings of the 5th IFIP International Workshop on Quality of Service (IWQoS'97), New York (1997) 37–48
46. Munson, J., Dewan, P.: Sync: A system for mobile collaborative applications. IEEE Computer **30** (1997) 59–66