Networking Protocols for Wireless Multimedia Streaming

by Amoolya Singh

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Anthony D. Joseph
Research Advisor

(May 19, 2000)

* * * * * * *

Professor Randy H. Katz
Second Reader

(May 19, 2000)

Contents

List of Figures

Abstract

The wireless link poses a significant challenge for sending video streams. This is due to the fact that current generation wireless links have low bit rate and high error rate compared to wire-line links. To send high bit rate delay-sensitive traffic over a wireless link, suitable video compression algorithms and transport/link protocols are needed. We built a wireless video system merging a low bit rate video codec with appropriate transport/link layer protocols that overcome the constraints imposed by the wireless link. In the general case, this project provides a general wireless network infrastructure to support real-time streaming applications. We assume that the application layer provides error resilience, rate control, and packetization functionalities to the network. In turn, the network layers allow error-resilient policies at the application to be reflected in the transport and link layers by enabling flexible checksumming schemes such as UDP Lite and PPP Lite. We evaluated the overall performance of this wireless video system using both quantitative performance metrics (such as throughput, jitter, packet loss, and end-to-end latency) and qualitative performance metrics (such as viewer perception of quality: smooth motion, changes in luminance, edge detection, etc).

1 Introduction

As cellular telecommunications and packet-switched data networks converge, ubiquitous computing applications [36] will enable users to access data "anywhere, anytime, anyplace." Web browsing on cellular phones is already a reality with the introduction of Nokia's and Ericsson's Wireless Access Protocol (WAP) phones [35] this year. Other multimedia services like mobile banking, voice-activated email, and cellular video are being prototyped in Europe, Asia and the US. These new applications require new functionality in the networking stack of Internet and cellular nodes as well as in the protocols that span these nodes.

Current network/system environments enforce ordered reliable delivery when in fact the application may be able to tolerate out-of-order, unreliable or corrupted data. As protection in the protocol stack is increased – in the form of forward error correction (FEC), checksums, or automatic repeat request (ARQ) mechanisms – there is a corresponding decrease in the effective data rate. This in turn decreases the ability of the application to react on the fly to transmission errors. Rather than over-engineering network protocols to anticipate every possible packet mis-ordering or corruption, we show that a protocol stack with flexible error correction schemes uses bandwidth more effectively, decreases retransmissions, and decreases latency and jitter. For example, if the application can stand bit error rates of 10^{-2} but requires in-order delivery, then it implicitly communicates those requirements to the network; the network then guarantees this level of error protection on a per-socket granularity. The application-level framing (ALF) model proposed by Clark and Tennenhouse [8] offers a powerful framework for enabling the application to determine what it considers "sensitive" data. One can imagine a graph of network protection versus application adaptivity; the balance point for each application is then determined by the notion of "utility" elaborated by Shenker and others in the QoS literature [7, 29].

1.1 Contributions of This Work

Research in this area has thus far focused on how to make delay-sensitive applications adaptive or error-resilient in the wired Internet. With the exception of the multicast community [4, 18], no one (to our knowledge) has developed a network architecture to enable multimedia streaming in low-bandwidth environments. The video coding community has extensively researched application-level error correction schemes [24, 27, 34, 37], but none at the transport and link layers. For an

in-depth discussion of related work, please refer to Section 6. To sum up, the contributions of this work are that

- we are exploring the very low bandwidth environment of wide-area wireless data networks;
- we are enabling real-time delay-sensitive multimedia flows by modifying error detection mechanisms in the network and not in the application; and
- we buttress simulation studies of error resilience with actual measurements on a wireless GSM[1] circuit-switched data network.

Specifically, our project aims to realize one aspect of the cellular/IP multimedia trend by prototyping "video over wireless" – we send video streams over a wireless interface and measure network performance and video quality. Challenges in implementing this system include the low bit rate (9.6 kb/s) and high bit error rate of the wireless channel. In addition, the sensitivity of human perception of multimedia [31] imposes tight requirements on jitter, delay, and image quality. In our system, the former set of challenges is addressed by changes in transport and link layer protocols, while the latter set is addressed by off-the-shelf error resilient video coding software.

2 Background

2.1 Application Layer

2.1.1 Video Transmission Systems

Generic video transmission systems consist of a video coding engine which encodes an analog video into a digital video bitstream, and a channel coding engine which transmits the bitstream over a (possibly lossy) channel. The former piece is commonly known as a **source coder** while the latter is referred to as a **transport coder**.

Source coders are themselves composed of two parts, waveform and entropy coders. The waveform coder transforms the original video using lossy techniques such as the Discrete Cosine Transform (DCT) and quantization. The motion estimation and compensation stage of the waveform coder predicts each frame from its neighboring frames, compresses the prediction parameters, and produces the prediction error frame. The prediction error stage codes the prediction and residual error in each frame.

The entropy coder then losslessly converts the output from the waveform coder into binary code words according to the statistical distribution of symbols to be coded. Common techniques used by the entropy coder are Huffman and arithmetic coding.

Finally, the transport coder performs channel coding, packetization and modulation, and transport level control to convert the video bitstream into suitable data units (packets) for transmission. The overall system is summarized in Figure 1. Note that although the transmission is typically only shown in the forward direction, systems with two-way communication channels (such as ours) are not uncommon. These will be detailed in the following section.

2.1.2 Error Detection and Concealment

Unfortunately, video transmission is not error-free. Bit errors and erasures are introduced at both the waveform and transport coders. There are three main mechanisms to detect these errors. First,

[1] Global System for Mobile Communications

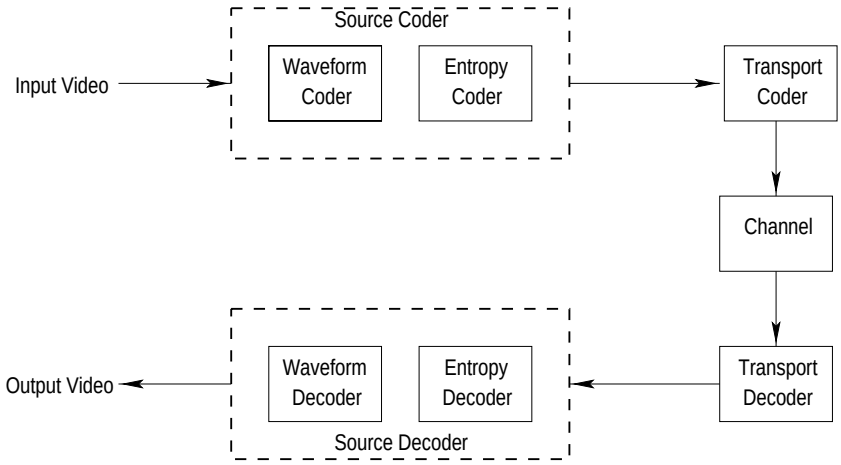


Figure 1: A Generic Digital Video Transmission System

transport-level detection involves adding header information or using FEC to prevent packet loss. Second, **video-level detection** involves using differences in pixel values (between adjacent lines, or between boundary values in a frame) to detect bit errors and erasures on a per-frame basis. The advantage of using transport-level detection is that transmission becomes more reliable; the downside is that including redundant information increases channel bandwidth, which may not be appropriate for low bitrate channels like wireless. Thus the advantage of using video-level detection is that it does not add bits to the stream beyond those allocated to the source coder. A third technique, known as **frequency domain detection**, compromises well between the first two techniques by using synchronized code words and incomplete variable length codes (VLCs) to detect bitstream errors.

Once errors have been detected, they must be concealed. This is typically achieved by adding "concealment redundancy" at the waveform, entropy, or transport coders. Given a video source model, total channel bandwidth, and channel error characteristics, one can design a source-transport encoder/decoder in three different ways.

First, **encoder-based** techniques are those in which the encoder plays the primary role in concealing errors. Examples include FEC, robust entropy/waveform coding, multiple description channels, layered coding, and joint source-channel coding. Second, **decoder-based** techniques are those in which the decoder plays the primary role in concealing errors, using estimation and interpolation. Examples include spatial and temporal smoothing, interpolation, and filtering. Finally, **interactive** techniques assume the presence of a backchannel from the decoder to the encoder; both parts of the system then cooperate to detect and conceal errors on the fly. Examples include ARQ, intelligently chosen retransmissions, and selective predictive coding with feedback from the decoder.

2.1.3 H.263+ Error Resilient Video Codec

The H.263 video codec standard was developed by the ITU for transmitting video streams at low data rates (< 64 kbit/s) over the public switched telephone network. It encodes low bit rate video streams using a motion-transform hybrid video coding framework.

The H.263+ standard (1998) improves upon the H.263 standard (1996) in the following ways: it

Figure 2: Sample Screenshots from H.263+ Video Bitstream

provides enhanced error resilience capabilities (appendices A-F from the standard), offers optional bitstream scalability, and enables better packetization with an underlying protocol such as RTP [5]. The H.263+ standard adds nine new features to the existing suite, including advanced intra coding, reduced block artifacts using a deblocking filter, reference picture selection and resampling, reduced-resolution updates, and modified quantization. The curious reader is referred to the voluminous ITU standard [12] for an exhaustive description of the codec. Two sample screenshots from the *"mom"* bitstream are shown in Figure 2. These screenshots are used as benchmarks to evaluate the performance of our video transmission system, detailed in Section 5.

2.2 Transport Layer

Real-Time Transport Protocol (RTP). RTP, the Real-time Transport Protocol [28], provides end-to-end delivery services for real-time data such as interactive audio and video. The protocol design allows application-level framing (ALF) and integrated layer processing. RTP provides extra information to the application layer in the form of sequence numbers, timestamping, payload type, and delivery monitoring. Depending on the particular real-time application at hand, RTP is either integrated with the application layer or the transport layer (UDP); both RTP and UDP contribute parts of the transport layer functionality.

It is important to note that RTP does not itself ensure timely delivery or other quality-of-service guarantees; it relies on lower-layer services to do so. RTP sequence numbers allow the receiver to reconstruct the sender's packet sequence, but are also used to determine the proper location of a packet (e.g. in video decoding) without necessarily decoding packets in sequence.

User Datagram Protocol (UDP). UDP is a connection-less unreliable best-effort transport layer protocol. At the transport layer of the protocol stack, we had the choice of using either TCP or UDP. TCP is the most common transport protocol for sending data. However, the main drawback of TCP is that it does not perform well when carrying delay-sensitive traffic such as audio or video. This is because TCP drops corrupted packets at the receiver, causing the sender to retransmit packets, which in turn introduces delay into the connection. An extensive body of research has examined modifications to TCP for real-time flows, including TCP-friendly applications [33], indirect TCP [2], etc. TCP-friendly modifications primarily address congestion control, with less attention to packet loss. Indirect TCP isolates mobility and wireless-related problems, but requires additional hardware in the network in the form of Mobility Support Routers.

In contrast, UDP does not require additional hardware nor does it retransmit lost packets. This guarantees immediate delivery. While an improvement over TCP, UDP is still not the ideal multimedia transport protocol because it drops corrupted packets, thus decreasing throughput.

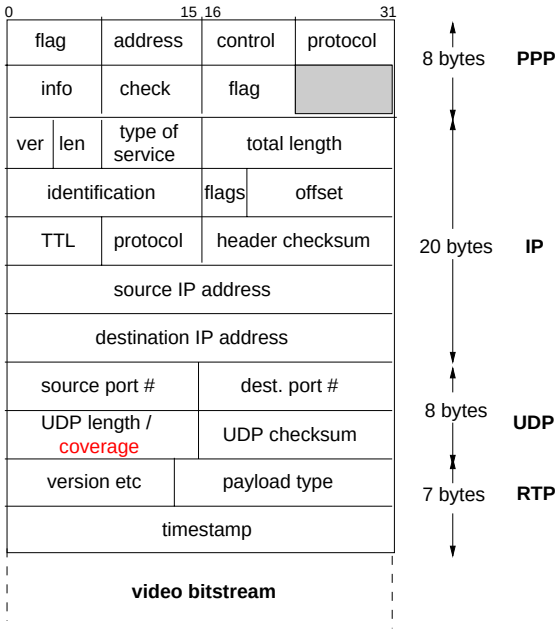


Figure 3: Packet Header for Video. Note the *coverage* field used by UDP Lite.

UDP Lite. The UDP Lite protocol [15] attempts to solve these problems by allowing the application to receive corrupted packets instead of dropping them altogether. This is achieved by a partial checksum which only covers a fixed amount of "sensitive" data. Integrating UDP Lite into existing UDP frameworks is simple: the *length* field in the UDP header is replaced by the *coverage* field (see Figure 3), which signifies how many bytes of the packet have been checksummed. With a checksum coverage value replacing the packet length, UDP Lite packets are treated like classic UDP packets with the checksum enabled.

The UDP Lite approach is better suited for video traffic where it is more important to receive frames at a constant rate even though some frames may be corrupted. By using UDP Lite as the underlying transport protocol, these applications can perform more robustly in the face of errors caused by the (wireless) network.

Socket Interface. We implemented our own version of a send/receive socket interface in C because existing implementations use TCP as the default transport-layer protocol [9, 32], whereas we needed to use RTP and UDP in the transport layer. In addition, we instrumented our socket interface to collect statistics as described in Section 3. The sender's socket sends data continuously while the receiver's socket blocks until it receives new data. This initially resulted in buffer overflows (and therefore packet drops) as the kernel buffer filled up faster than the radio link layer could empty [17]. The trade-off in this design was thus between latency and packet loss rate: increasing the kernel's buffer resulted in fewer dropped packets but worsened the end-to-end delay, while increasing the rate at which packets were sent resulted in less end-to-end delay but higher packet loss rates. Using standard rate control calculations, we determined the balance point of the socket buffer to be 8192 B (up from the default 4096 B).

2.3 Link Layer

2.3.1 Data Link

Point to Point Protocol (PPP). PPP [30] is a data link protocol for point-to-point links. It has two main functions: encapsulation of IP datagrams, and error detection using a polynomial Frame Check Sequence (FCS).

PPP Lite. The implementation of UDP Lite obtained from Lulea University included modifications to the device drivers to implement a "lite" version of PPP for Unix FreeBSD. Keith Sklower ported these modifications to BSDi with additional functionality for simply ignoring PPP checksums in the receiver. This effectively limits the checksumming to that provided by UDP Lite, which can be controlled at the socket level.

2.3.2 Radio Link

The GSM standard allows data traffic in two modes. Non-transparent mode runs a semi-reliable protocol, the Radio Link Protocol, described below. Transparent mode, described in the paragraphs following, simply utilizes the GSM radio link without embedding any additional functionality.

Radio Link Protocol (RLP). The GSM Radio Link Protocol [1] is a full-duplex reliable link layer protocol derived from the HDLC logical link layer protocol. Fixed-size data frames of 30 bytes (with 6 bytes of header information) are sent/received every 20 ms; the overall bandwidth is therefore 1200 bytes/s or 9.6 kbit/s. All frames are strictly aligned to the GSM radio block size used as a basis for channel coding. RLP uses two mechanisms for error recovery: *selective reject* and *checkpoint recovery*.

Figure 4 illustrates the selective reject mechanism, which is explicitly initiated by the receiver to request retransmission of missing frames. Whenever a frame is received out of order, the receiver sends an "SREJ" control frame specifying the sequence number of the missing frame. The sender is not required to "roll back," but instead retransmits the frame of the sequence number equal to the sequence number sent by the receiver. The sender then returns to its state before retransmission and continues to transmit frames. It should be noted that every SREJ frame is protected by a single retransmission timer and the sender is limited to retransmitting the same frame no more than N2 (typically 6) times.

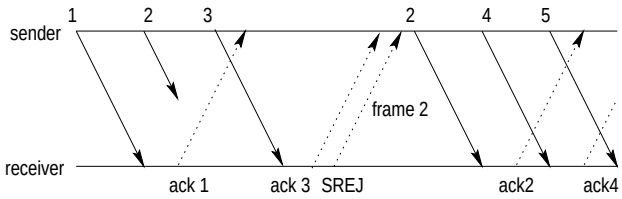


Figure 4: Selective Reject in RLP

Figure 5 illustrates the checkpointing mechanism, which is initiated by the sender whenever there is a timeout of an acknowledgment. The sender sends a "Poll" control frame and waits for the receiver's response. The receiver responds with a control frame indicating the receive sequence number of the frame that it is expecting. Finally, the sender retransmits all the frames in sequence, starting from the first frame before the timeout.

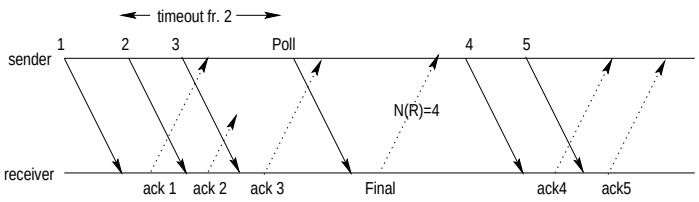


Figure 5: Checkpointing in RLP

The ARQ mechanism of RLP will introduce delay that may not be tolerable when sending video streams. GSM allows us to choose between transparent mode (without ARQ) and non-transparent mode (with ARQ). Turning on RLP increases reliability but also increases delay. On the other hand, turning off RLP decreases delay but also decreases reliability. By experimenting with these two modes, we better understood the role of RLP in overall performance.

GSM non-RLP Mode. Since transparent mode uses the GSM radio link without any additional functionality, it allows consistent delay on the data transfer. Other protocols for compression (e.g., V.42bis, MNP5) and error control (e.g., MNP4, V.42) may be run as separate protocols by the two communicating modems. Data compression yields data rates approaching the 64 kbits/s of the GSM base network.[2] In addition, transparent mode is typically run asynchronously, where data is not clocked but includes start/stop bits for each byte and some additional overhead. The other option is to run transparent mode synchronously, where clocking is achieved with the use of an additional semi-reliable protocol. In this case, bandwidth overhead is almost as high as that of RLP.

Radio Link Tradeoffs. To understand the tradeoffs between RLP and the various non-RLP configurations, we performed a set of ping measurements and collected round-trip-time (RTT) and packet loss statistics. These results are detailed in Appendix B, and show that non-RLP without error control and compression yields the minimum end-to-end delay and overhead.

3 Design Overview

We now present a brief overview of the layers in our network stack and the challenges and solutions posed at each layer. Figure 6 illustrates the different parts of our design. Our setup consists of a mobile host which communicates with a fixed host[3] through a GSM [19, 25] circuit-switched connection.

The mobile host communicates with the GSM Network using an Ericsson mobile phone and a commercial Ericsson wireless modem.[4] Commands to initialize the modem are included in Appendix C. Going end to end, we have RTP with either UDP or UDP Lite[5] running at the transport layer, IP on the network layer, and PPP at the data link layer. The wireless link from the mobile host to the GSM base station runs either RLP or non-RLP.

[2]However, it should be noted that most compression algorithms perform worse when fed already-compressed data; this is what we observed when trying to transmit compressed video bitstreams.

[3]Both hosts are running UNIX BSDi 3.0.

[4]DC23 PCMCIA card.

[5]UDP Lite always runs over PPP Lite.

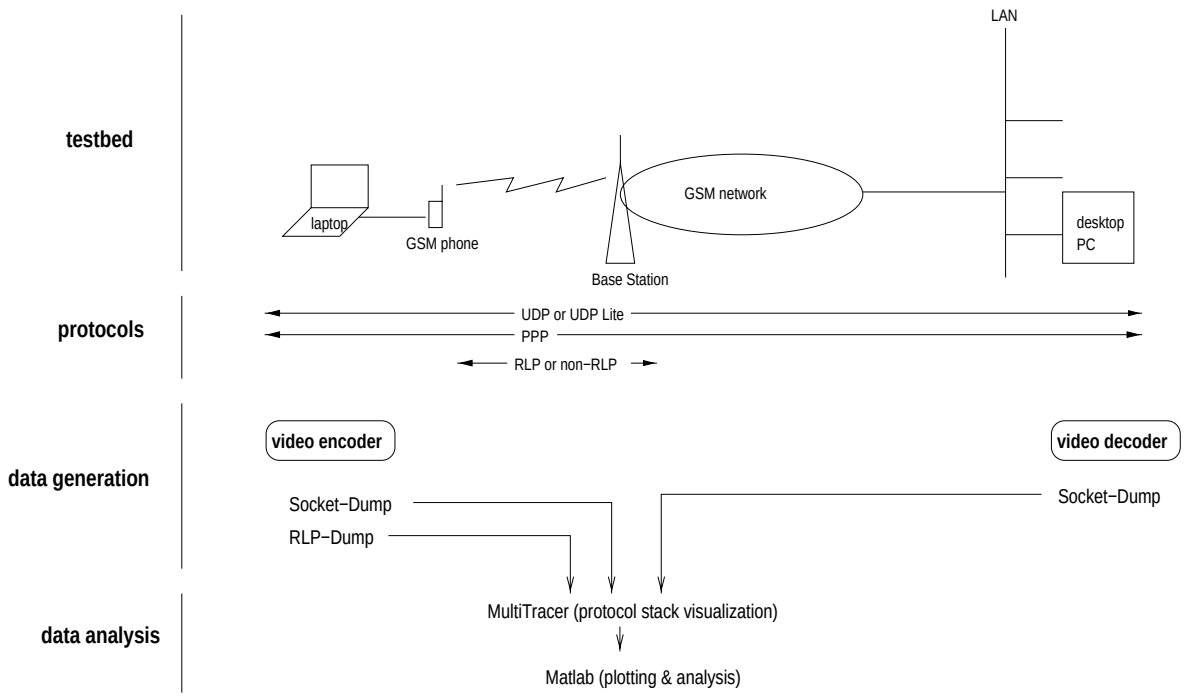


Figure 6: Cellular IP Testbed

At the sender (mobile host), video streams were encoded using the H.263+ standard error-resilient video codec for low bit rates. The resulting encoded bitstreams, with a QCIF resolution of 176 x 144 pixels, bit rate of 10 kbits/s and frame rate of 10 Hz, were packetized into RTP packets using a recommended packetization algorithm [38], then fed into a socket connection. At the receiver (fixed host), RTP packets were read off the socket, stripped of headers, and reassembled into a contiguous stream which was read by the decoder in real time. The resulting video was displayed on the receiver and analyzed qualitatively. The protocol stack with appropriate interfaces is shown in Figure 7.

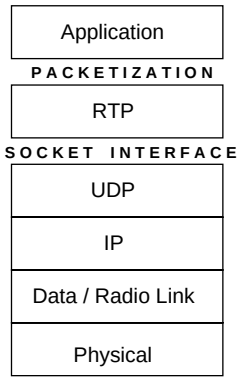


Figure 7: Networking Stack

To trace information at the RTP/UDP sender and receiver, we developed *socket-dump*, which generates a single log file containing timestamps and byte numbers specifying when a packet was

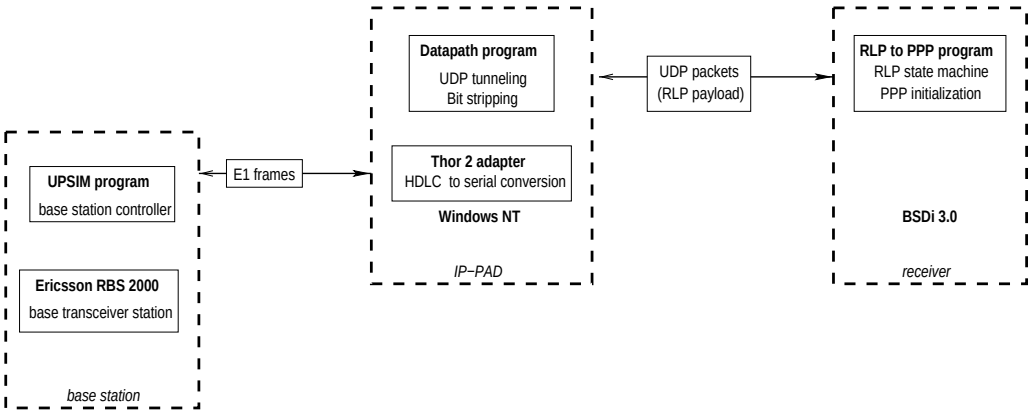


Figure 8: Iceberg GSM Subsystem

placed in the sender's buffer. From the data generated by *socket-dump*, we graphed time/sequence plots and calculated statistics. In particular, we were interested in analyzing end-to-end delay, packet inter-arrival time, throughput information and also some exceptional conditions (e.g. retransmissions, detailed in Section 2.3). To visualize the data generated by *socket-dump* and *RLP-dump*, we used *MultiTracer* [17], a set of Perl scripts that convert the trace data into the input format required by plotting tools such as Matlab and *xgraph*.

4 Implementation

4.1 Iceberg Testbed

The Iceberg (Internet-based Core for CEllular networks BEyond the thiRd Generation) project at UC Berkeley [13] aims to integrate diverse access networks with Internet, telephony, and data service capabilities. Service-intensive, network-based real-time applications are deployed by securely embedding computational resources in the switching fabric. As a result, we will understand how to encapsulate existing application services (e.g. speech-to-text, location-awareness). We will also be able to provision "all-IP" cellular networks for scalability, multinetwork mobility, and security support while studying QoS issues for delay-sensitive flows.

The GSM subsystem of our experimental testbed is the focus for integrating a GSM base station with an IP core network. Details of the "Iceberg GSM network" cloud from Figure 6 are shown here in Figure 8. The various pieces of equipment are described in Raman's MS thesis [26], pp. 16-19.

Briefly, a data call is set up as follows. After initializing the base station to accept data calls on a particular timeslot, the Datapath program is started up on the IP-Packet Assembler and Disassembler (IP-PAD). This polls the E1 interface card, which is an HDLC adapter with incoming T1/E1 lines.[6] When the mobile host (not pictured here) calls a phone number, the controller machinery at the User Part Simulator (Upsim) receives these signaling frames. At the IP-PAD, GSM data frames arrive on the E1 line. The Datapath program pulls them off, strips the necessary bits in the header (see Appendix A for the format of a data frame), demultiplexes the frame, and extracts the data payload from each of the relevant timeslots. This payload is then tunneled

[6] Thor-2 Dual T1/E1 Adapter, Odin TeleSystems [14].

through UDP to a Unix BSDi workstation for further processing. At the Unix workstation, the RLP-to-PPP program starts the RLP state machine to process incoming packets, then invokes `pppattach` to start a PPP connection between the sender and receiver. At this point, the sender and receiver can communicate just as though there was a wired network connection between them.

4.2 Lessons Learned

As of August 1999, a BSDi UNIX implementation of RLP by Bela Raythoni (Ericsson) was working correctly on hosts initiating wireless data transfer, but was not tested on the "receive" side. Although the setup was sufficient if the user was simply interested in controlling the "send" side of data transfer, it was not sufficient if the user was interested in controlling both ends of data transfer, or obtaining any end-to-end statistics about RLP, or using the mobile host as the receiver in a data call. To realize this, it was necessary for the Iceberg IWF host to run RLP correctly, analogous to what happens inside a commercial GSM network.

Pacific Bell Testbed
RLP code $\mapsto$ laptop's COM port $\mapsto$ cell phone $\mapsto$ PacBell GSM network $\mapsto$ PacBell IWF $\mapsto$ PSTN $\mapsto$ receiver modem

Iceberg Testbed
RLP code $\mapsto$ laptop's COM port $\mapsto$ cell phone $\mapsto$ Iceberg base station $\mapsto$ IP-PAD "IWF" (Thor board + RLP code) $\mapsto$ LAN

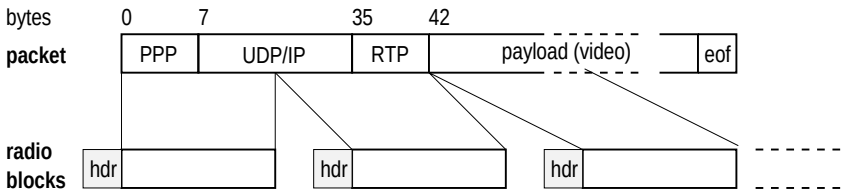
Figure 9: Pacific Bell and Iceberg Testbeds

Data calls initiated through the base station's Upsim controller were being torn down by the controller before any actual data transfer could begin. This was because the Upsim software was assigning timeslots incorrectly. Once that problem was fixed, data calls stayed up but no data was actually being received on the IP-PAD (acting as the Iceberg IWF). Keith Sklower and I wrote a program which tested Raythoni's RLP implementation by simulating a duplex send/receive wireless channel. We found that the sender was expecting an acknowledgment to its XID request, sent as the very first frame. (XIDs are used in a single handshake to periodically negotiate control information.) The receiver was not correctly set up to send acknowledgments, so after N2 (6) re-transmissions of its XID request every 48 frames, the sender's RLP state machine simply dropped the connection. We corrected the receiver to send acknowledgments, and also increased its window size to twice that of the sender to prevent XID collisions. This resulted in both sender and receiver working correctly.

In retrospect, although the conceptual problem was not very intricate, debugging roughly 10,000 lines of completely undocumented code – accompanied only by opaque ETSI specifications – was challenging in its own right.

Further work involved plugging in the corrected RLP implementation into the IP-PAD software, and testing data calls from end to end. This involved the additional step of modifying the IP-PAD IWF software to remove all dependencies on the Upsim, in anticipation of the shift to an Abisco interface.[7]

[7]The Ericsson TSS 2000 "Abisco" (Abis Communicator) will replace the Upsim as the base station controller. In addition to all the functionality provided by the Upsim, the Abisco also provides a Smile interpreter, runtime inter-process message database, and a Windows interface for functional and test debugging.



Fragmentation is used as a basis for wireless packet simulation. Note that the "header" is not really 2 bytes long as shown in scale: there are two bits of overhead for each byte in asynchronous mode, which we simplify as 2 bytes of overhead per 28 bytes of data.

Figure 10: Radio Block Fragmentation

When the corrected RLP implementation was installed on the IP-PAD, further problems arose. Data calls could be set up and torn down, but were unstable; various minor bug fixes (e.g., replacing an idle polling loop with a timed poll to reduce processor cycles, testing the timing of the RLP state machine, etc.) did not rectify this. We guessed that the clocking mechanisms on Windows NT were not fine-grained enough to support the delay-sensitive RLP code, but could not migrate to a Unix BSDi platform (where the RLP code was known to work) because the drivers for the E1 interface card were written only for Windows. Therefore we devised an intermediate solution in which the IP-PAD would simply strip the headers from incoming data frames, and tunnel the payload (using UDP) to a Unix BSDi workstation, which would terminate RLP. This solution worked well, and we could set up and maintain data calls for extended periods of time; however, the round trip times from sender to receiver ranged from 1200 to 1800 ms, about twice as high as the RTTs on the PacBell testbed, and clearly unacceptable for taking measurements!

We noticed an unusually high number of retransmissions in RLP with the new UDP tunneling set up. Since the UDP connection between the IP-PAD and the BSDi workstation was on a 10Mb/s shared Ethernet, we surmised that there were many packet losses, in turn triggering radio link-level retransmissions. To test this hypothesis, we conducted several runs of 1000 `ping`s between various machines on the Ethernet at various times of the day. The results were contrary to what we expected: there were no packet losses. Other debugging approaches including checking for memory leaks and profiling the memory usage of the IP-PAD software provided no insights. At the time of this writing, the problem has not been investigated further: we think the retransmissions might be caused by slight differences in clock cycles between the IP-Pad and the BSDi workstation.

The approach most likely to succeed would be to port the Thor2 device driver code to Unix, test it aggressively, and run both RLP and its attendant device driver software on the *same* machine.

5 Performance Evaluation

We conducted both simulations and experiments on wireless video transmissions. Although we had planned to conduct experiments on the Iceberg testbed, we were not able to take realistic measurements owing to the high RTT described in Section 5. Therefore we used the commercial Pacific Bell GSM network instead. The simulations served the purpose of isolating the error resilience effects of various codecs as well as understanding the viability of UDP Lite as a protocol for wireless multimedia streaming.[8]

[8]Simulations also helped keep the PacBell phone bills at a minimum!

5.1 Channel Simulation

Performance studies of multimedia applications over wireless networks require intense analysis of real video transmission over an existing network infrastructure. In addition to being very time-consuming, it is oftentimes difficult to isolate specific problems. To understand the impact of high bit error rate wireless channels on multimedia applications, we designed a wireless simulator, $WSim$ which takes advantage of wireless error traces collected in previous work [16]. $WSim$ (see Figure 11) takes as input a video stream, a wireless error trace, and a protocol configuration. It uses these to generate an output video stream with some corruption.

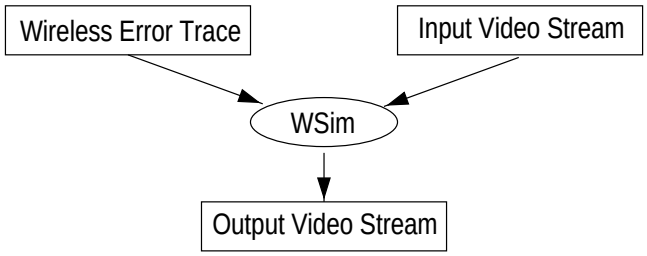


Figure 11: $WSim$ block diagram

The wireless error traces we used are specific to the particular FEC and interleaving schemes implemented in GSM circuit-switched data connections. A trace is comprised of a binary sequence where each element represents the state of a radio block[9] in a GSM wireless channel. There are two states. A *one* represents a corrupted radio block, while a *zero* represents a correct radio block. The BLock Error Rate (BLER) defines the average rate of corrupted radio blocks.

The traces were collected in different channel environments such as stationary, walking and driving. Traces collected in stationary environments are either termed *bad* or *good* depending on the signal strength at the mobile phone [16]. The error trace specifies if a 30-byte radio block is corrupted or not. However, we did not know the distribution of errors *within* a block. We initially use a simplistic approximation by randomly corrupting 20% of the bytes in the block. Towards the end of this project, we collected byte-level traces using the `tip` command, which simply wrote bytes between two wirelessly connected devices. In future work, we plan to feed this byte-level error distribution into the channel model.

$WSim$ is based on 215 minutes of traces collected in a bad stationary environment. We simulate two protocol configurations:

- UDP running over transparent (non-RLP) mode
- UDP-Lite running over transparent mode

The algorithm for WSim is as follows:

1. Choose ADU_size = size of the application data unit (ADU) in bytes.
2. Choose transport protocol (UDP or UDP-Lite).
3. Repeat following steps until end-of input video stream is reached.
   (a) Read ADU_size bytes from input video stream.

[9] In current circuit-switched GSM systems, a radio block contains 30 bytes.

UDP UDP Lite

Figure 12: Simulation Screenshots

(b) Generate *packet_size* in bytes: add RTP/UDP/IP/PPP header size (7+8+20+7=42 bytes, see Figure 3) to *ADU_size*.

(c) Generate N = number of radio blocks to carry *packet_size* bytes of data, where each radio block contains 28 bytes of data (see Figure 10).

(d) Repeat following steps N times.

- Read radio block from input wireless error trace
- If the radio block is corrupted, corrupt 28 bytes of data.

(e) If the chosen transport protocol is UDP

- if packet is corrupted, drop the packet i.e. do nothing
- else write *ADU* into output video stream

(f) If the chosen transport protocol is UDP-Lite

- if packet RTP/UDP/IP/PPP header is corrupted, drop the packet i.e. do nothing
- else write *ADU* into output video stream

Our goal was to compare the two protocol configurations and determine which one offers better video quality. We ran *WSim* on the *"mom"* video stream using a wireless error trace of 1.58% BLER. Figure 12 shows representative screenshots of video streams produced by *Wsim* for both UDP and UDP Lite. Observe that the video quality with UDP is far worse than with UDP Lite owing to the fact that UDP drops corrupted packets while UDP Lite transmits them.

5.2 Experimental Results

For our experiments, we ran video over wireless for the four possible combinations of transport and radio link protocols: UDP and UDP Lite with RLP; UDP and UDP Lite with non-RLP.

The tradeoff here is thus between (1) a connection-less best-effort datagram protocol in the transport layer, with and without flexible checksumming, and (2) a wireless link layer protocol with and without reliability. Running UDP or UDP Lite over RLP means that drops at the transport layer might trigger retransmissions in the radio link machinery. This negates the effect of using a best-effort datagram service in the first place.

To "showcase" the difference between UDP and UDP Lite, we turn off retransmissions and aggressive checksumming in lower protocol layers. This is achieved by running the wireless link in transparent (non-RLP) mode, with error correction and compression turned off. The experiments with non-RLP therefore pinpoint differences between UDP and UDP Lite.

We will refer to each video transmission, lasting four minutes, as a *run*. We collected a total of 4480 minutes of wireless video traces, corresponding to 1120 runs. Each protocol combination was

repeated for ten runs for both good and bad channel conditions. Thus, each data point on each graph presented below corresponds to 40 minutes of transmission.

We define a good channel condition as one in which the signal strength varied from 15 to 20 dB[10] corresponding to a readout of 4-5 bars on the mobile phone's indicator, while a bad channel condition was one in which the signal strength varied from 4 to 9 dB corresponding to a readout of 1-2 bars on the mobile phone's indicator.

For each run, we collected various statistics as described in Section 2 and used these to compute four key metrics:

- End-to-end delay (s)
- Jitter, also known as packet inter-arrival time (s)
- Packet loss (% of packets lost)
- Throughput (kbits/s)

In the following several graphs, we present results for 160 minutes of the collected traces. The mean value for each metric and protocol combination is plotted as a single point, with Y error bars denoting standard deviation.

5.2.1 Delay

The delay of the connection from end to end is plotted in Figure 13. Notice that the mean delay of the non-RLP connection, for both UDP and UDP Lite, is smaller than the mean delay of the RLP connection. This is as expected: a reliable link-layer protocol like RLP will introduce end-to-end delay due to retransmissions at the link layer, even though the overlying transport protocol (UDP or UDP Lite) is only best-effort.

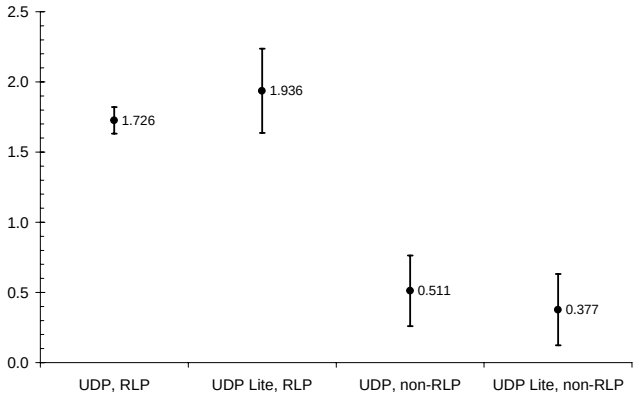


Figure 13: End-to-End Delay (seconds)

[10]Obtained by periodically querying the wireless modem using the command `at+csq`.

5.2.2 Jitter

The jitter or packet inter-arrival time is plotted in Figure 14. The main result in this graph is that packet inter-arrival time is remarkably constant across all combinations of transport and link-layer protocols. Again comparing RLP with non-RLP, we notice that the range of inter-arrival times is larger for RLP (introduced by re-transmissions) than for non-RLP. We believe that the difference in standard deviations across all four protocol combinations can be explained by slight variations in the wireless channel environment rather than by any significant protocol effects.

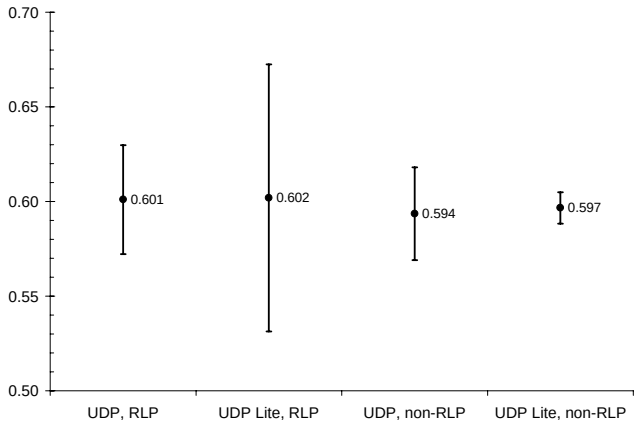


Figure 14: Jitter or Packet Inter-Arrival Time (seconds). Note that the Y-axis does not begin at zero.

5.2.3 Packet Loss

Packet loss is plotted as a percentage of packet drops in Figure 15. These results are simple to understand. Loss is 0% for the RLP connection, because the radio link protocol uses ARQ to retransmit lost frames or acknowledgments. Loss is slightly higher for the non-RLP connection because there is no reliability at either the link or transport layers. As expected, packet loss is much lower for UDP Lite than UDP (1.1% vs. 2.1%), because the checksumming function in UDP Lite is turned off.

5.2.4 Throughput

The final metric of interest is throughput, presented in Figure 16. Since packet loss is significantly lower for RLP than non-RLP, we expect to see higher throughput for the RLP connections, and this is indeed the case. Zooming in on the non-RLP connections, we notice that the throughput for UDP is markedly lower than that of UDP Lite. This is explained by the fact that UDP has higher packet loss than UDP Lite, which in turns drives its throughput down.

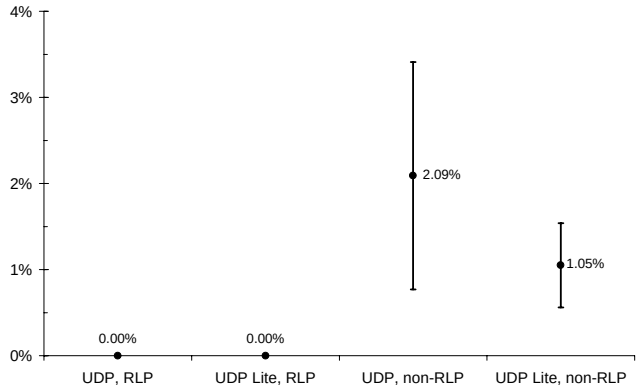


Figure 15: Packet Loss (%)

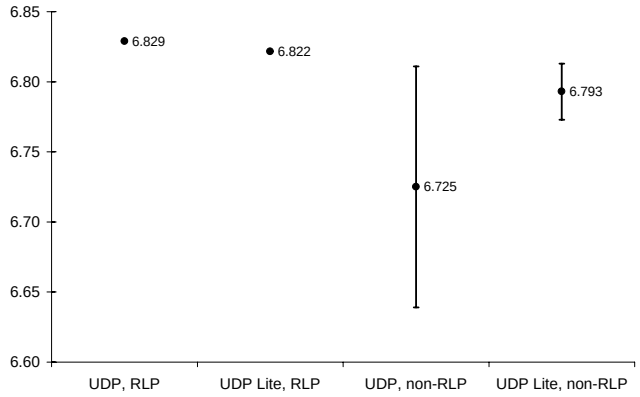
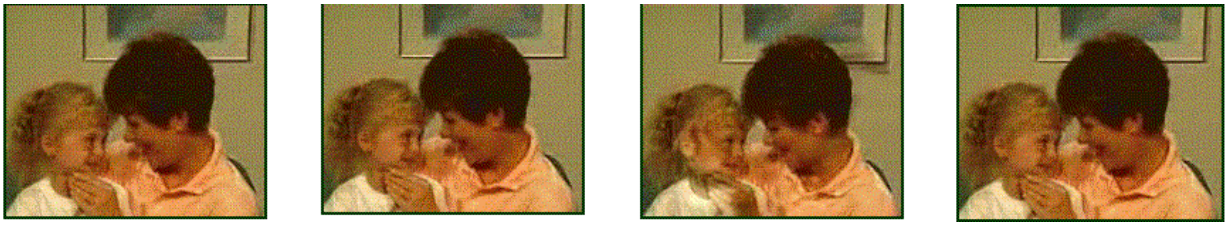


Figure 16: Throughput (kbits/s). Note that the Y-axis does not begin at zero.

5.2.5 Perceived Video Quality

Delay, jitter, and packet loss all affect perceived video quality. High delay causes a long lag between when an image is encoded/sent and when it is received. High variation in packet inter-arrival times causes the video to appear "shaky" or jerky. Finally, packet loss causes the image to be blurred or severely distorted. The pictures shown in Figure 17 are samples from the H.263+ video bitstream sent over varying protocol combinations. They are ordered from left to right as the x-axes on the four preceding plots. From these screenshots, we see that RLP (the two left pictures) causes excellent video quality. However, the real-time delay caused by RLP (appreciably high, as described



UDP/RLP UDP Lite/RLP UDP/non-RLP UDP Lite/non-RLP

Figure 17: Experimental Screenshots

previously) cannot be depicted in a screenshot. On the other hand, non-RLP (the two pictures on the right) causes slightly worse video quality. When using non-RLP, it is preferable to use UDP Lite over UDP: notice that the UDP screenshot shows a large amount of distortion near the girl's ear (due to packet losses) while the UDP Lite screenshot does not.

5.3 Discussion

In summary, UDP Lite (with non-RLP) provides 26% less end-to-end delay, constant jitter, slightly higher (1%) throughput, 50 % less packet loss, and significantly better video quality than UDP. These results allow us to comment on what protocol combinations may be used for particular applications. For example, a delay-sensitive application that does not have stringent error-free requirements might perform better using the error resilience of UDP Lite and the low delay characteristics of non-RLP. On the other hand, an interactive error-intolerant application would perform better using the high reliability of RLP offset by the high throughput of UDP Lite.

These tradeoffs are summarized in the table below.

Type of Application	**Example**	**Protocol Choice**
intolerant & rigid*	batch *email, ftp*	TCP/RLP
	interactive *telnet, web*	UDP/RLP
tolerant & adaptive*	hard real-time *wb*	UDP/RLP
	adaptive real-time *vic, vat*	UDP Lite/ non-RLP

*Adapted from [7].

6 Related Work

Several efforts similar to this research exist both in industry and academia. Here we will outline the most prominent and attempt to compare and contrast them with our work.

PacketVideo Corporation [23] based in San Diego, CA develops software for streaming video to mobile devices wirelessly. First demonstrated in November 1999, their patented video codec plays MPEG-4 video streams on the application-specific integrated circuits used in mobile phones.

However, since their technology is patented, performance is somewhat clouded by market hype and analyses are not publicly available. That said, PacketVideo's impressive trade show demos and product offerings have justifiably gotten intense press coverage.

NEC Corp. [20] in September 1999 announced a prototype of a two-piece third generation cellular video handset which transmits images and sound simultaneously. The phone and viewer are connected using Bluetooth short distance radio technology while the video is compressed using the MPG-4 standard. NTT DoCoMo [10] has also produced a prototype video phone similar to NEC's. However, these prototypes are aimed at the W-CDMA (wideband code division multiple access) network system, supporting data rates of 64 Kb/s and 128 Kb/s. The high bandwidths distinguish NEC's and NTT's products from our research. Also, with GSM standards taking and more of wireless market share (71% in 1998), it is unclear how CDMA or AMPS technology can compete.

On the academic side of the fence, researchers in Sweden have profiled the performance of multimedia streams over network nodes running UDP Lite in the kernel [15]. Their simulations of real-time network traffic (with and without partial checksums on the transport and link layers) show that the decision to turn checksumming on or off must be a function of the particular network environment rather than a function of the particular multimedia stream. This decision concurs with our results.

Wendi Heinzelman has developed unequal error protection schemes for MPEG-4 [22] video which adapts the level of correction across a packet to the importance of the corresponding bits [6]. Her work is distinguished from ours in that her application-specific communication protocols for wireless networks focus mainly on energy-efficient cluster-based routing and media access.

Closer to home, Avideh Zakhor's group at Berkeley has developed hierarchical FEC, subband coding, and other video compression techniques in the context of their Matching Pursuits [21] video codec. They are also examining TCP-based bandwidth-scalable techniques; these can be grouped with the TCP-friendly body of work.

Finally, the Comet group at Columbia University [11] is examining adaptive mobile networking issues through *Mobiware*, a CORBA- and Java-based networking middleware toolkit which adapts to time-varying network conditions. Although the thrust of their work sounds quite similar to the ideas presented here, the primary difference is in (important) details: all their experiments are based on local-area (2 Mb/s WaveLAN) and ATM networks, and not the wide-area IP-based Internet. The extremely high bandwidth of their air interface implies a significantly different set of research problems than ours.

7 Conclusion

In this report, we have argued for application-level adaptation schemes and against aggressive network-level protection of multimedia traffic over wireless. We base this argument on the ALF model and on the implementation of flexible checksumming schemes such as UDP Lite and PPP Lite, which allow the application to determine what data it considers "sensitive" to corruption or loss. Through simulation of the wireless channel and experiments transferring live video over a radio interface, we have shown that a networking stack instantiated with these "lite" protocols offers markedly less delay, less packet loss, and higher throughput than a traditional protocol stack, while keeping the jitter and video quality constant. In addition, implementing these protocols is not difficult: UDP Lite involves changes to only a few hundred lines of kernel code, and at the user level, it is possible to invoke UDP Lite on a per-socket basis. As video coding technology such as MPEG-4 improves and all-IP cellular networks such as GPRS [3] mushroom, we will be able to

build rate-adaptive systems which utilize the radio backchannel for network-level feedback. This simplistic but powerful networking approach, when combined with appropriate video codecs, will enable truly error-resilient mobile multimedia communication.

8 Future Work

There are several possible extensions to this work. At the application layer, one might incorporate unequal protection schemes which would work in concert with UDP Lite. Also, we were initially interested in prototyping video transmission for both H.263+ and MPEG-4 video codecs; however, the MPEG-4 code[11] was extremely difficult to work with. After a lengthy and painful compilation process, we were still not able to encode or decode video streams in real time due to undocumented configuration parameters and untraceable floating-point errors.

At the socket layer, a better implementation might entail a more dynamic feedback mechanism between the application and link layers. By providing real-time feedback on radio channel conditions, the encoding application can adapt its transmission rate accordingly.

In the wireless simulator, we plan to collect byte-level wireless error traces which we will feed into *WSim* as the channel coding model.

Acknowledgments

Much of this work was done in collaboration with Almudena Konrad, without whose persistence, attention to detail, and positive spirit we would still be slogging through performance measurements. I am indebted to Profs. Anthony Joseph and Randy Katz for their encouragement, guidance and feedback. Reiner Ludwig, Bela Raythoni, and Stephan Baucke, all of Ericsson, were a ready wealth of technical knowledge. Last but not least, I thank Keith Sklower, the resident Unix guru, for all manner of systems-related programming and for his patient lessons on kernel debugging.

References

[1] ETSI GSM Technical Specification 04.22. GSM Radio Link Protocol for data and telematic services, Dec 1995.

[2] A. V. Bakre and B. R. Badrinath. Implementation and Performance Evaluation of Indirect TCP. *IEEE Transactions on Computers*, 46(3):260–278, March 1997.

[3] C. Bettstetter, H.-J. Vögel, and J. Eberspächer. GSM Phase 2+ General Packet Radio Service GPRS: Architecture, Protocols, and Air Interface. *IEEE Communications Surveys*, 2(3):2–14, 1999.

[4] J.-C. Bolot, T. Turletti, and I. Wakeman. Scalable Feedback Control for Multicast Video Distribution in the Internet. In *Proceedings of ACM SIGCOMM*, 1994.

[5] C. Bormann, L. Cline, G. Deisher, T. Gardos, C. Maciocco, D. Newell, J. Ott, G. SUllivan, S. Wenger, and C. Zhu. RTP Payload Format for the 1998 Version of ITU-T Rec. H.263 Video (H.263+). *RFC 2429*, Oct 1998.

[11] Version 4.0 of the Fine-Grain Scalability Software, downloaded from CCETT, France

[6] M. Budagavi, W. R. Heinzelman, J. Webb, and R. Talluri. Wireless MPEG-4 Video Communication on DSP Chips. *IEEE Signal Processing Magazine*, January 2000.

[7] D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *Proceedings of ACM SIGCOMM*, 1992.

[8] D. Clark and D. Tennenhouse. Architectural Consideration for a New Generation of Protocols. In *Proceedings of ACM SIGCOMM*, 1990.

[9] D. Comer. *Internetworking with TCP/IP. Vol 1: Principles, Protocols, and Architecture.* Prentice Hall, 1991.

[10] M. Takano *et al.* DoCoMo's Exhibits for Telecom '99. *NTT DoCoMo Technical Journal*, 1(3):32–40, Mar 2000.

[11] Comet Group. `comet.columbia.edu/mobiware/overview.html`.

[12] ITU-T Recommendation H.263. Video coding for low bit rate communication, Feb 1998.

[13] Iceberg Project. `iceberg.cs.berkeley.edu`.

[14] Odin TeleSystems Inc. The Thor-2 T1/E1 Adapter. `www.odints.com`.

[15] L.A. Larzon, M. Degermark, and S. Pink. UDP Lite for Real-Time Multimedia Applications. *Proceedings of the Sixth IEEE International Workshop on Mobile Multimedia Communications*, 1999.

[16] R. Ludwig, A. Konrad, and A. Joseph. Optimizing the End-to-End Performance of Reliable Wireless Links. In *Proceedings of ACM Mobicom*, 1999.

[17] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, and A. Joseph. Multi-Layer Tracing of TCP over a Reliable Wireless Link. In *Proceedings of ACM SIGMETRICS*, 1999.

[18] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-Driven Layered Multicast. In *Proceedings of ACM SIGCOMM*, 1996.

[19] M. Mouly and M. B. Pautet. *The GSM System for Mobile Communications.* Cell and Sys, France 1992.

[20] NEC Corp., Japan. `www.nec.com`.

[21] R. Neff and A. Zakhor. Very low bit rate video coding based on matching pursuits. *IEEE Transactions on Circuits and Systems for Video Technology*, 7.1:158–171, 1997.

[22] MPEG4 Overview. `drogo.cselt.stet.it/mpeg/ standards/mpeg-4/mpeg-4.htm`.

[23] PacketVideo Corp., San Diego, CA. `www.packetvideo.com`.

[24] R. Puri, K. Ramchandran, and A. Ortega. Joint source channel coding with hybrid FEC/ARQ for buffer constrained video transmission. In *Proceedings of Multimedia Signal Processing*, 1998.

[25] M. Rahnema. Overview of the GSM System and Protocol Architecture. *IEEE Communications Magazine*, pages 92–100, April 1993.

[26] B. Raman. Personal Mobility in the Iceberg Integrated Communication Network. Technical Report UCB/CSD-99-1048, University of California at Berkeley, May 1999. www.cs.berkeley.edu/~bhaskar/ms/tech-rep.ps.

[27] R. Rejaie, M. Handley, and D. Estrin. Quality Adaptation for Congestion Controlled Video Playback over the Internet. In *Proceedings of ACM SIGCOMM*, 1999.

[28] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. *RFC 1889*, Dec 1995.

[29] S. Shenker. Fundamental Design Issues for the Future Internet. *IEEE Journal on Selected Areas in Communications*, 13:7, Sep 1995.

[30] W. Simpson. The point-to-point protocol. *RFC 1661*, Jul 1994.

[31] R. Steinmetz and C. Engler. Human perception of media synchronization. Technical Report 43.9310, IBM European Networking Center, Oct. 1993.

[32] W.R. Stevens. *TCP/IP Illustrated Vol 2: The Implementation.* Addison-Wesley, 1995.

[33] The TCP Friendly Web Page. www.psc.edu/networking/tcp-friendly.html.

[34] Y. Wang and Q.-F. Zhu. Error Control and Concealment for Video Communication: A Review. *Proceedings of the IEEE*, 86(5):974–997, May 1998.

[35] WAP Forum. *Wireless Application Protocol*, 1999. www.wapforum.org.

[36] M. Weiser. The computer for the 21st century. *Scientific American*, 263(3):94–104, 1991.

[37] Q. Zhang and S. A. Kassam. Hybrid ARQ with Selective Combining for Video Transmission over Wireless Channels. In *Proceedings of IEEE International Conference on Image Processing*, 1997.

[38] C. Zhu. RTP Payload Format for H.263 Video Streams. *RFC 2190*, Sep 1997.

A Bit Format of a GSM Data Frame

The C bits are control bits which carry signaling and state information.

	Bit number							
Octet no.	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	1	C1	C2	C3	C4	C5	C6	C7
3	C8	C9	C10	C11	C12	C13	C14	C15
4	1							
5	1							
6	1							
7	1							
8	1	data quadrant 1 (72 bits)						
9	1							
10	1							
11	1							
12	1							
13	1							
14	1							
15	1							
16	1							
17	1	data quadrant 2						
18	1							
19	1							
20	1							
21	1							
22	1							
23	1							
24	1							
25	1							
26	1	data quadrant 3						
27	1							
28	1							
29	1							
30	1							
31	1							
32	1							
33	1							
34	1							
35	1	data quadrant 4						
36	1							
37	1							
38	1							
39	1							

B Ping Statistics

For each modem configuration we send 100 *pings* with payload of 500 bytes.
Ping command: `ping -c 100 -s 500`

Mode	**Synch-ronous?**	**Data Compr.**	**Error Ctl.**	**RTT** (ms)	**Loss** (%)
RLP	Y	N/A	N/A	2023	0
non-RLP	N	MNP 5	MNP 4	4425	25
	N	none	MNP 4	3670	25
	N	none	none	1576	33

Notes:
It is not possible to run RLP in asynchronous mode.
The fourth non-RLP case, viz. data compression without error control, is not supported by the modem.

C Modem Initialization

```
% tip tty02                use tip to connect to device modem
connected
atz                        initialize modem
OK
at+cbst=7,0,0              9600 bps, asynchronous, no RLP
OK
at+es=0,0,0                turn off error correction
OK
at+ds=0                    turn off data compression
OK
at+er=1                    report error correction policy used
OK
at+dr=1                    report data correction policy used
OK
at+csq                     check signal strength on phone

+CSQ: 16,99                (low 4 - 10, med 10 - 15, high >15)

OK

atdt6429514                dial modem number

+ER: NONE                  no error correction being used

+DR: NONE                  no data compression being used


CONNECT 9600

BSDI BSD/OS 3.0
(akira.CS.Berkeley.EDU)
login:
```