An Efficient End-Host Protocol Processing Architecture for Real-Time Audio and Video Traffic

Khawar M. Zuberi and Kang G. Shin
Real-Time Computing Laboratory, The University of Michigan
{*zuberi,kgshin*}*@eecs.umich.edu*

Abstract

Information appliances (IAs) are mass-produced, low-cost devices for specialized computation/communication tasks. Examples include webTV, smart cellular phones, and web phones. Since low cost and low power-consumption are important requirements in the design of these devices, IAs use relatively slow/cheap CPUs (usually 20–40 MHz). Real-time audio and video communication over the Internet is an integral part of many IAs which means that despite slow hardware, the communication subsystem within the OS must be able to efficiently handle heavy network traffic. This paper presents a protocol architecture which reduces I-cache miss overheads (benefiting short audio messages) and also enables use of single-copy without any hardware support or restrictions on network APIs (benefiting long video messages). We implemented UDP/IP to evaluate our architecture and measurements show that overheads for short messages are reduced about 20% while overheads for long messages are reduced 15–22%.

1 Introduction

Information appliances (IAs) [1] are single-user devices with Internet connectivity, used for specialized communication and information retrieval purposes. IAs include devices such as webTVs, smart cellular phones with e-mail, PDAs, and web video phones. With annual production volume of IAs expected to reach 48 million units by year 2001 [2], IAs are becoming an important class of computation devices.

Since IAs are mass produced, keeping per-unit costs low is a primary design objective. As a result, IAs use simple, low-cost hardware. For example, the current generation of *personal information managers* (PIMs) use processors running at 16–44 MHz [1]. Since audio/video communication is a primary function performed by IAs, the communication subsystem within the OS must be highly efficient to work well with the low-cost, slow hardware of IAs for both short audio and long video messages. Different overheads come into play depending on whether short or long messages are being processed. *Data-touching* overheads (which include data copying and checksum overheads) tend to dominate when dealing with long messages. For short (audio) messages, copying overheads are not important, but messages are sent once every 10–30ms [3]. With messages arriving with such high frequency, *non-data-touching* overheads (context switching, interrupt handling, I-cache miss overheads, etc.) become an important part of protocol processing.

Studies have shown that receive-side protocol processing is more complicated and has higher overhead than the send-side [4, 5] and this is what limits throughput; so, here we focus on improving receive-side overhead. For reducing non-data-touching overheads, we present *layer bypass* which uses application-specific knowledge to safely bypass select layers within the protocol stack, completely avoiding all I-cache misses associated with those layers. Regarding data-touching overheads, we exploit the periodic nature of video applications. We show that the single-copy scheme presented for non-real-time systems in [6] — which requires specialized network adapter hardware to be feasible for non-real-time systems — works well without any hardware support for multimedia applications because of their periodic nature.

For evaluation, we implemented UDP/IP using our protocol architecture within the EMERALDS real-time operating system [7]. EMERALDS is designed for use in small embedded systems such as digital cellular phones. We chose UDP as the protocol to implement since it is commonly used for audio and video applications.

The next section gives an overview of protocol processing overheads. Section 3 presents our schemes for reducing these overheads which are then evaluated in Section 4. The paper concludes with Section 5.

2 Protocol Architecture Issues

Following is an overview of I-cache and data-copying overheads, schemes proposed by other researchers to reduce these overheads, and shortcomings of these schemes.

2.1 Efficient I-Cache Usage

Communication protocols are designed to accommodate varying communication patterns and error conditions. As a result, a large portion of the protocol code is devoted to checking for rarely-occurring errors or special message formats. These checks are usually coded as shown in Figure 1. Most of the time, there are no errors so that the bodies of the `if` statements never execute. However, code is still fetched into the I-cache, causing replacement misses. Moreover, repeated branches can cause CPU pipeline stalls. For relatively slow CPUs such as those used in IAs, this results in significant non-data-touching overhead. Researchers have proposed techniques such as *outlining/cloning* [8] and *incremental specialization* [9] to reduce I-cache misses, but these schemes involve low-level optimization of frequently-executed code in the

```
if (check1) { ... }
if (check2) { ... }
if (check3) { ... }
```

Figure 1: Typical structure of error checks in protocol code.

protocol stack, and this entails considerable effort on part of the programmer. This indicates a need to develop an easier-to-use scheme to reduce I-cache misses.

2.2 Single-Copy Architectures

The single-copy network architecture was proposed by network adapter (NA) designers [6] to reduce data-touching overheads. The idea is to design a NA with enough buffer space so that on transmission, data is copied once from user-space directly to the NA, while on reception, data stays in the NA until the application makes a receive system call and then data is copied directly to user-space.

To work well for general-purpose systems, single-copy schemes require the NA to have "flexible" buffers. The Afterburner NA [6] uses linked lists to manage NA buffers, so it can continue to receive packets as long as some free buffers are available. This is a more complicated and expensive NA design than common NAs such as LANCE which uses circular queues of buffers. For reception, LANCE fills buffers in the receive ring until it reaches a filled buffer at which point it starts dropping packets (even if other buffers are free down the ring). But, as we will show later, for real-time applications, the single-copy architecture is feasible *when combined with a real-time task scheduler*, and it works even with cheap NAs such as LANCE (which is essential to keep costs low in IAs).

3 Protocol Architecture

The next subsection describes the basic structure we chose for our protocol architecture, followed by a description of our protocol processing optimizations.

3.1 Basic Structure

We chose *lazy receiver processing* (LRP) [10] (Figure 2) as our basic architecture. Under LRP, the packet filter [11] tries to forward real-time packets directly to queues associated with the destination thread where packets stay unprocessed until the application makes a receive system call. This is possible for real-time messages since the application threads are usually periodic so that packets are always processed within a known time interval. Non-real-time packets are forwarded to a special network thread which performs protocol processing and keeps the message until the final destination thread makes a receive call. LRP minimizes *priority inversion* [12] and also saves one context switch for real-time messages compared to architectures which always use intermediate network threads for protocol processing, which is why we use LRP.

Next, we describe our optimizations for protocol processing. Note that these optimizations do not depend on LRP and would work equally well with other protocol architectures.

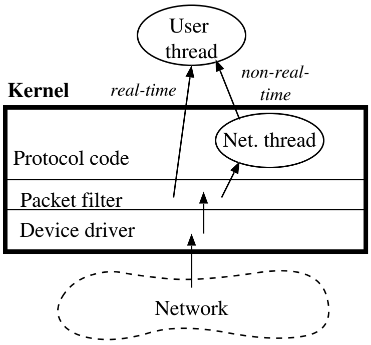

Figure 2: Lazy receiver processing.

3.2 Reducing Non-Data-Touching Overheads

For lowering non-data-touching overheads — especially those related to I-cache misses — we present a new scheme called *layer bypass*. It is easier to apply than low-level optimizations like outlining and specialization and does not require any in-depth analysis of protocol code.

Layer bypass relies on the observation that most of the functionality implemented by various protocol layers is simply not needed when processing short messages. The few operations that are needed are either already duplicated in the packet filter or can be easily migrated there. This allows various protocol layers to be bypassed, completely avoiding all I-cache misses these layers may have caused.

Layer bypass can be applied as follows. When writing the packet filter, the protocol specification for various protocol layers is first studied to determine which aspects of the protocol are not needed for a given message stream (such as live audio messages). Those layers are identified which perform little or no functions. The few useful operations these layers do perform can all be placed together in the packet filter (or some other such small module appropriately inserted in the protocol stack). Then the filter is programmed to detect messages belonging to certain message streams (like audio) and bypass the redundant layers for these messages. For example, if the IP layer is to be bypassed, the filter will forward packets straight to the transport layer.

Layer bypass can be used very effectively for audio messages. UDP/IP is commonly used for such messages, so first consider the IP layer which performs the following major functions:

IP: Routing, fragmentation/reassembly, IP address checks, IP header checksum, checks for malformed headers and packets, IP option processing.

A receiving host does not perform any routing. Short live audio messages need not be fragmented/reassembled. IP options are used for network testing, so they do not apply here. The packet filter checks the destination IP address, so this function of the IP layer is already being handled by the packet filter. The IP header checksum and other error checks can be bypassed safely for the following reason: the packet filter examines the various fields in the header to detect audio packets. If the header has been corrupted, the filter will not recognize the packet; it will be forwarded to the IP layer, normal processing will occur, the errors will be detected, and the packet will be discarded.

Now we look at the feasibility of bypassing the UDP layer which performs the following major functions:

UDP: UDP datagram checksum, port address checks, generate ICMP messages if destination port does not exist, pass incoming datagrams to correct socket.

The UDP checksum is usually turned off for live audio messages. The packet filter already checks the destination port address. If this check fails, the packet is routed through the full protocol stack where error handling (if needed) can occur. So, for live audio messages, the packet filter bypasses both IP and UDP layers and forwards packets straight to the socket layer.

Layer bypass can also be used for handling other types of short messages such as web server requests as discussed in [13].

3.3 Improving Data-Touching Overheads

We now show that the single-copy scheme — which, without hardware support, has limited value for non-real-time systems — can be used effectively for video communication with no special hardware support. The key to the effectiveness of the single-copy scheme in real-time systems is a real-time scheduler which guarantees that the application executes at its period and does not face unpredictable delays. Hence, incoming packets will stay in the NA buffers for no longer than the period of the application. This is in contrast to non-real-time applications where no such bound exists on how long an application may take to retrieve its packets from the NA.

Real-time audio and video applications run with some period T. T for audio is quite short, usually 10–30ms [3], but video applications can run as slow as 30 or even 10 frames/s, giving a T as large as 0.1s. This is the maximum time messages for a video application have to stay in NA buffers. If NA buffers are about to overflow and the video messages have not been processed, packets for these messages have to be copied out of the NA into kernel buffers to make room for incoming packets. This can occur if a burst of non-real-time packets arrive, filling NA buffers in a short period of time. Following is an analysis of how often this might happen when both real-time and non-real-time packets are being received through the NA.

Estimating Non-Real-Time Packet Arrivals: Poisson processes have previously been used to model packet arrivals. However, studies have shown that wide-area network traffic is too bursty to be correctly modeled by a Poisson process [14]. In fact, only empirical models exist for web browsing [15] and other mass data transfer applications. This precludes any closed-form derivation of non-real-time packet arrival distributions. Instead, we present an engineering approximation of packet arrival rates to see if the single-copy scheme can be used successfully in IAs.

We use web browsing as a representative non-real-time networking application. Measurements of web traffic have shown that retrieval of even small web pages take more than 2 seconds [15]. This is the time needed to look up the remote host's DNS entry and establish TCP connections. After this initial phase data transfer begins at the rate of 1 byte per 90–100μs [15]. Most web pages are relatively small-sized. Considering the small display screens that IAs have, we assume a 10kB page size and 20 packets to carry 10kB. With these assumptions and using 90μs as the per-byte transfer time (which is faster than the wireless link speeds available to most IAs today), we get a packet arrival rate of 22 packets/s or 2.2 packets/0.1s. Even if due to burstiness, five times as many packets arrive within $T = 0.1s$, we still get only 11 packets/T. Even if 5 bursts of 11 packets/T occur during download (highly unlikely since the majority of web pages fit in fewer than 30 packets) and the user spends just one second reading the document, the probability of getting a burst of 11 packets/T is only 5 times in $35T$ seconds or 0.143. This is negligible considering that NAs typically have 128–256 buffers.

Note that the above analysis is true even for the NAs with the simplest buffer-management policies, such as LANCE. As such, the single-copy scheme for real-time applications does not require any special/expensive hardware support which is an important consideration for IAs.

4 Evaluation Results

We want to evaluate the effectiveness of our architecture in handling both short audio and long video messages. We implemented our protocol architecture within EMERALDS on a 25MHz Motorola 68040 processor. The 68040 is typical of CPUs used in many IAs today. (Refer to [13] for evaluation results on a faster processor.) We use two 68040s in our experiments, connected by a 10Mb/s private Ethernet using the LANCE network adapter.

For evaluation, we implemented UDP/IP using our architecture. The protocol and LANCE device driver code was taken from FreeBSD 4.4 and minor modifications were made to make it work with EMERALDS. For simplicity, we used a UDP/IP-specific packet filter. Interested readers are referred to [11] for more generalized high-performance packet filters.

4.1 Performance Improvements

We sent datagram messages from one processor to another and measured the total overhead of receive-side protocol processing including interrupt handling and all relevant context switches using a 5MHz on-chip timer. For each data point (fixed message size) we repeated this experiment 100 times and averaged the results. Further increase in number of samples did not result in any significant change in averaged results.

Short Messages: Figure 3 shows the total receive overhead for short message sizes. This figure presents measurements for the cases when processing is done by a special network thread (labeled "standard architecture"), regular LRP, and LRP when layer bypass is

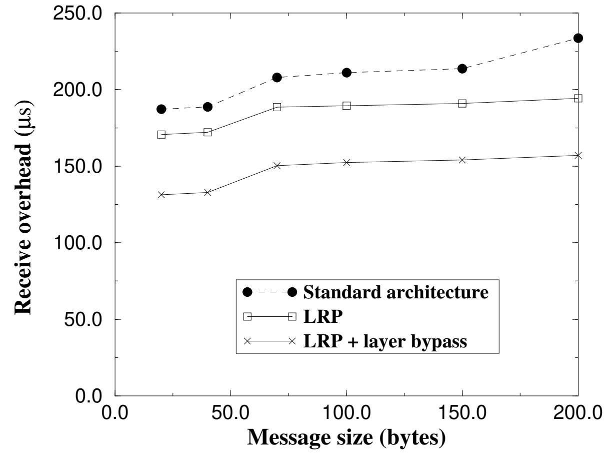

Figure 3: Receive overhead for short messages.

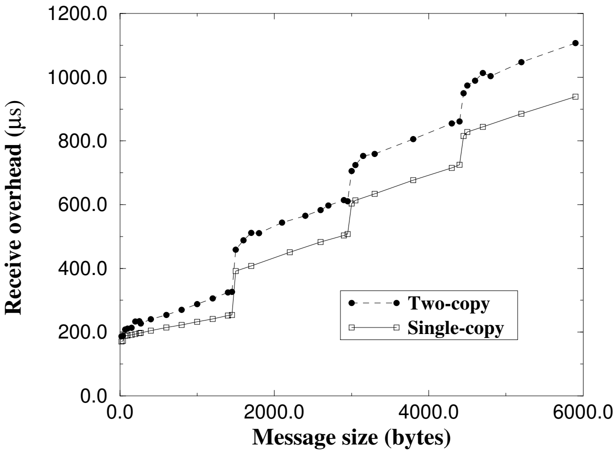

Figure 4: Receive overheads for long messages.

used as well. In all cases, the UDP checksum is turned off. From the figure, we see the benefit of bypassing IP and UDP layers. Performance is improved 20% (beyond that of LRP). Note that the sharper variations in the plots are a result of BSD's mbuf allocation scheme [4] and are not related to the protocol architecture.

Long Messages: Figure 4 plots the receive overhead for messages ranging from 20 to 6000 bytes. It shows that single-copy incurs 15–22% less overhead than the two-copy scheme. Note that the sharp increases in overheads approximately every 1500 bytes is due to messages being fragmented into Ethernet packets, each of which generates a separate interrupt.

5 Conclusion

Information appliances (IAs) are an emerging class of devices which are used for specialized communication tasks such as audio/video communication over the Internet. The Internet-centric nature of IAs combined with the need to keep costs low in these mass-produced devices (which results in the use of slow/cheap processors) dictates that the communication subsystem in the OS be highly efficient to enable audio/video communication despite slow hardware. In this paper we presented a protocol architecture which improves message reception overhead for real-time audio and video communication. Overhead for short live audio messages is reduced 20% by safely bypassing protocol layers. For long video messages, we showed that the single-copy scheme can be used effectively — without hardware support — to reduce overheads by 15–22%.

References

[1] T. Lewis, "Information appliances: Gadget netopia," *IEEE Computer*, vol. 31, no. 1, pp. 59–70, January 1998.

[2] *Appliance war could make web less open.* News Briefs, IEEE Computer, vol. 30, no. 10, pp. 20–25, October 1997.

[3] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," RFC 1890, January 1996.

[4] J. Kay and J. Pasquale, "Measurement, analysis, and improvement of UDP/IP throughput for the DECstation 5000," in *Proc. Winter USENIX*, pp. 249–258, January 1993.

[5] D. Kandlur, D. Saha, and M. Willebeek-LeMair, "Protocol architecture for multimedia applications over ATM networks," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1349–1359, September 1996.

[6] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley, "Afterburner," *IEEE Network*, vol. 7, no. 4, pp. 36–43, July 1993.

[7] K. M. Zuberi and K. G. Shin, "EMERALDS: A microkernel for embedded real-time systems," in *Proc. Real-Time Technology and Applications Symposium*, pp. 241–249, June 1996.

[8] D. Mosberger, L. L. Peterson, P. G. Bridges, and S. O'Malley, "Analysis of techniques to improve protocol processing latency," in *Proc. SIGCOMM*, pp. 73–84, August 1996.

[9] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole, and K. Zhang, "Optimistic incremental specialization: Streamlining a commercial operating system," in *Proc. Symposium on Operating Systems Principles*, pp. 314–324, December 1995.

[10] P. Druschel and G. Banga, "Lazy receiver processing (LRP): A network subsystem architecture for server systems," in *Proc. Operating Systems Design and Implementation*, October 1996.

[11] D. Engler and M. F. Kaashoek, "DPF: Fast, flexible message demultiplexing using dynamic code generation," in *Proc. SIGCOMM*, pp. 53–59, August 1996.

[12] C. Mercer and H. Tokuda, "An evaluation of priority consistency in protocol architectures," in *Proc. IEEE Conf. Local Computer Networks*, pp. 386–398, October 1991.

[13] K. M. Zuberi, *Real-Time Operating System Services for Networked Embedded Systems*, PhD thesis, University of Michigan, EECS Dept., 1998.

[14] V. Paxson and S. Floyd, "Wide area traffic: The failure of poisson modeling," *IEEE/ACM Trans. Networking*, vol. 3, no. 3, pp. 226–244, June 1995.

[15] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of WWW client-based traces," Technical Report BU-CS-95-010, Boston University, Computer Science Department, 1995.