Improving the Efficiency of Remote Resource Usage in Distributed Real-Time Systems

Paul S. Usher and Neil C. Audsley

Real-Time Systems Research Group
Computer Science Department
University of York
York YO10 5DD

October 14, 2004

Abstract

Typical real-time operating systems (RTOS), such as RT-Linux support distribution by adding network capabilities (eg. network protocol stack) without any architectural change. This approach maybe appropriate for systems that have sufficient resources to meet all of an applications functional and non-functional requirements, but for low-resource systems such an approach often imposes a significant overhead – applications wishing to access remote hardware resources cannot do so without significant RTOS involvement at the local and remote nodes.

This paper outlines an approach being taken towards more efficient remote resource access. Fundamentally, the approach intends to minimize the software overhead incurred when communicating with remote resources by building RTOS functionality upon a distribution layer.

1 Introduction

Network sensors and mobile phones have shown that useful devices need not be physically or computationally large in order to be useful. In fact the increasing pace of development in these areas seems to have only fueled further interest in ubiquitous and pervasive computing. Such devices may be limited in computational terms, but they are intended to interconnect in a seamless fashion in order to undertake tasks that would be beyond them individually. To achieve this they may need to access remote resources. This is illustrated by considering a low resource device that is able to provide a limited function in isolation, but if granted access to remote computational power or resources it is then able to deliver improved functionality.

In such an environment it is increasingly advantageous if there is no distinction between local and remote resources, i.e. Remote resources are accessed and used in the same manner as local resources. To illustrate this consider memory access in multiprocessor (e.g. SMP) or distributed shared memory systems [CDK01]. In such systems accessing remote memory uses the same mechanisms as local memory, noting that contentions have to be managed. For other resources, the same easy mechanism is not available. Usually, the access is via a remote network operation via network protocol stacks, imposing a heavy overhead. Clearly, this has implications upon a real-time system's ability to meet timing requirements.

This paper outlines the problems that are evident when remote resource access is achieved via a conventional RTOS and network stack. Then, the paper outlines an approach that is currently being taken to reduce this overhead. The context of the paper is that of general-

purpose real-time operating systems, such as RT-Linux.

2 Current Model

Access to remote resources in current systems is generally quite difficult and Figure 1 illustrates how this is typically achieved. In this illustration the application transmits a request via the network stack of its OS to the proxy process, which is executing the following operations in a loop:

1. Use a system call to wait until an application requests a service.
2. Use a system call to interact with the resource on behalf of the application.
3. Wrap the results of the interaction up in a message.
4. Use a system call to send the message back to the client application.

What this illustration does not show is that typically the remote resource would be mapped into the file system of the applications local OS. This allows the application to interact with the resource using normal systems calls, whilst the operating system generates the request message before using the network stack to transmit the message.

The main point of note from this diagram is that all access to such resources - even those that are low-level - must be affected via a message sent to some form of proxy. It also illustrates that any such access causes a sizable number of context switches to occur on the remote machine as well as requiring 4 traversals of the network stack and 2 send operations on the connection medium.

2.1 Medium performance

At first sight the biggest barrier to improving the performance of remote resource access functionality is going to be the communications medium. However recent studies show that network bandwidth is increasing at a faster rate than micro-processor performance [Mar02].

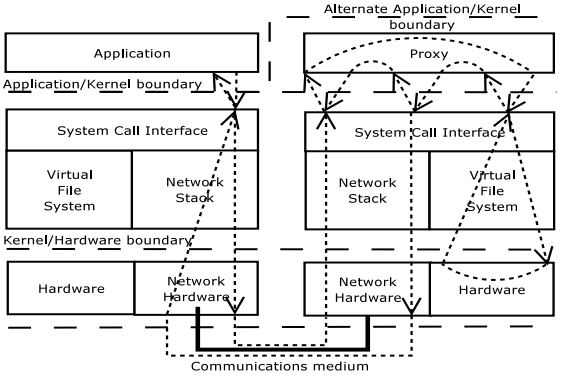

Figure 1: Current model of remote resource access

Unfortunately the bandwidth of the computers main memory has not increased with equal speed and hence it can be a factor that limits performance. One way to address this problem is Remote DMA - RDMA -, where network devices are able to transfer blocks of memory to another machine without the intervention of the CPU, thereby reducing the amount of work the CPU has to carry out [Amm04].

2.2 Network stack performance

In many cases reliable message delivery will be achieved via TCP/IP. Unfortunately the gap between TCP/IP performance and processing speed is increasing, with the throughput of the TCP protocol seeing around 60-80% of the performance improvement seen by the CPU. Worse still, higher level applications can see a considerably lower level of improvement with HTML benchmarks indicating that the latency seen when loading HTML pages has decreased by only 30% in real terms [Mar02].

The primary reason for this lack of improvement is that the architectural improvements made in modern micro-processors favor data or code that is repeatedly accessed [CS00, Mar02]. Unfortunately the architecture of the network stack minimizes accesses to the same data - especially when using zero-copy protocols - which limits any possible improvements.

It has also been shown that the processing overheads involved with this architecture are

a significant barrier to performance, with the cost of checksumming and copying the data occupying only 22% of the senders time and only 16.7% of the receivers [CS00].

2.3 Context switch performance

TCP/IP is not alone in getting little performance gain from modern CPUs as many operating systems have also seen little improvement. There are several possible reasons for this, but one of the most interesting is that switching contexts - user to kernel, or vice-versa - has become increasingly expensive due to the architecture of the CPU.

These context switches occur in Figure 1 whenever a message crosses one of the boundaries. As you can see there can be a significant number of them when using this model. To address this problem some systems utilize proxies that are resident in the kernel, thereby reducing such overheads. The affect of this kind of change is hard to quantify, but it is worth bearing in mind that the relative performance of any context switch has worsened quite dramatically in the last decade.

In the early 1990's this performance was increasing by 50-80% of the throughput increase seen by the CPU. Unfortunately by the end of the decade this figure has fallen to 12-28% [Ous90, Mar02]. Such operations are still faster in absolute terms, but relatively speaking operating system calls are now more expensive.

2.4 Software architecture

Current operating systems where designed under the assumption that local machines have sufficient resources to satisfy all of an applications requirements. As a consequence the network stack through which any remote access must occur, was only ever intended to be used for accessing other applications. Consequently the system lacks the necessary functionality to efficiently access resources in a manner that is independent of the current location.

Systems such as LINUX do of course allow remote access to a file system via local file systems which acts as a proxy [BC02]. But how many operating systems allow remote access to the device containing such a file system, and how many of those allow the local driver to be bypassed so that physical access to the device can be achieved. In addition some things such as memory and CPU time are not considered to be resources and hence no remote access to them is permitted.

3 Towards Efficient Remote Resource Access

To address the issues raised in the previous sections, an approach is being investigated that builds RTOS functionality upon a distribution layer which provides efficient access to remote resources. A two-fold approach is being taken, whereby the overheads of the network stack and virtual file system are reduced as much as possible. These issues are discussed further in the following sections.

3.1 Reducing Virtual File System Overhead

In Figure 1 the Virtual File System (VFS) provides the mechanisms by which an application can name and access a resource (local or remote). The VFS is inherently layered, with functionality broken down into smaller components. To do this it traps information within the component so that interaction is only possible via the published interfaces. Unfortunately this hiding of information can also be a barrier to performance, since some of the information that is trapped in a layer could be used by upper layers to make optimizations.

Currently, work is progressing to reduce the VFS overhead by allowing access from a remote node directly to the device drivers. This is illustrated in Figure 2, where remote access to resources at a low level is achieved without involving the complete VFS functionality (as shown in Figure 1).

3.2 Reducing Network Stack Overhead

TCP/IP communication between applications on the same machine is limited by the overheads of the protocol itself rather than the performance of the CPU or the systems memory [Mar02]. Initial tests indicate that local communication via TCP/IP sockets achieve only 11% of the throughput seen be AF_UNIX sockets, whilst AF_UNIX sockets incur only 47% of the overheads seen by their TCP/IP counterparts.

To improve on this level of performance it is proposed to allow negotiation between the layers so that additional information can be pushed to the upper layers, thereby permitting alternate actions in order to improve performance. This flow of additional information would therefore have a tendency to make the layered architecture flatter by removing unnecessary parts of a protocol in particular situations. Flatter communication hierarchies are likely to incur less runtime overheads, but they could also be harder to produce. In addition to which they may be less flexible, but since the optimization was performed on a per client basis rather than globally no flexibility is sacrificed.

These flexible communication protocols should allow operating systems to negotiate a low level network communications protocol that allows applications to more efficiently access remote resources by removing much of the overhead of TCP/IP style stacks.

In such a system remote access to a resource might look something like that shown in Figure 2. This illustrates that the receiving operating system has been able to optimize its network communication so that it can access the resource on behalf of the client without requiring all of the processing of the TCP/IP stack. This kind of flexibility potentially provides promising performance benefits for communication between clients, especially with regards to the reduction of communication latency and protocol processing overheads.

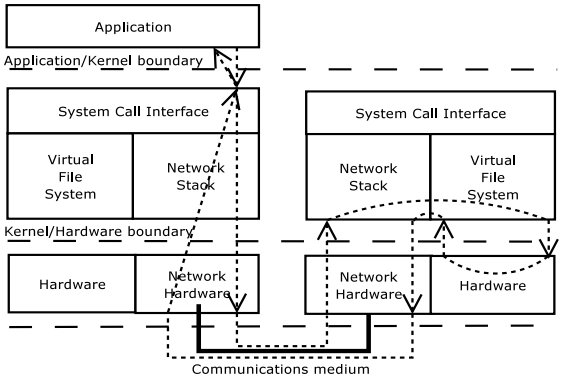

Figure 2: New model of remote resource access

References

[Amm04] Understanding remote direct memory access (RDMA). Technical report, Ammasso Inc, 2004.

[BC02] Daniel P. Bovet and Marco Cesati. *Understanding the LINUX KERNEL*. O'Reilly & Associates, Inc., second edition, 2002.

[CDK01] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems Concpets and Design*. Addison-Wesley Longman Publishing Co., Inc., third edition, 2001.

[CS00] Guo Chuanxiong and Zheng Shaoren. Analysis and evaluation of the TCP/IP protocol stack of linux. In *International Conference on Communication Technology*, volume 1, pages 444–453, 2000.

[Mar02] Evangelos P. Markatos. Speeding up TCP/IP: Faster processors are not enough. In *International Performance Computing and Communication Conference*. IEEE, 2002.

[Ous90] John K. Ousterhout. Why aren't operating systems getting faster as fast as hardware? In *USENIX Summer Conference*, pages 247–256, 1990.