14

A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass architecture

Y.H. Thia (*)[1] and C.M. Woodside (**)

Newbridge Networks, Inc., Ottawa, Canada (*) and
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada (**)

Abstract — The Reduced Operation Protocol Engine (ROPE) presented here offloads critical functions of a multiple-layer protocol stack, based on the "bypass concept" of a fast path for data transfer. The motivation for identifying this separate processing path is that it involves only a small subset of the complete protocol, which can then be implemented in hardware. Multiple-layer bypass also eliminates some inter-layer operations such as queue and buffer management, context switching and movement of data across layers, all of which are a significant overhead. ROPE is intended to support high-speed bulk data transfer. The paper describes the design of a ROPE chip for the OSI Session and Transport layer protocols, using VHDL. The design is practical in terms of chip complexity and area, using current gate array technology, and simulation shows that it can support a data rate approaching 1 gigabit per second, in a connection attached to an end-system.

Keyword codes: C.2.2, B.4.1
Keywords: Network Protocols, Data Communications Devices

1 Introduction

The advent of Fibre Optic technology, which offers high bandwidth and low bit error rates, has shifted the performance bottleneck from the communications channel to the communications processing in the end-points of the system [26]. Other trends such as improved quality-of-service guarantees will reinforce this effect. The heavy processing load is due to a combination of operating system overhead, protocol complexity, and per-octet processing on the data stream. To alleviate the end-system bottleneck one may consider new protocols [10], improved software implementation of existing protocols [5, 35], parallel processing techniques [14, 21, 38], special protocol structures [15, 30] and hardware assist [22] by offloading all or part of the protocol functions to an adaptor. This paper takes the latter approach.

The key problems associated with offboard processing include:

- Partitioning the functionality between the host and the adaptor is difficult and may easily lead to a complex additional protocol between the two parts, which may cancel out or offset the potential gain from offloading. For example, the buffer management task [36] may be offloaded, but this leaves the problem of control for accessing it within the full protocol logic.

[1] This research was done while Dr. Thia was at Carleton University

G. Neufeld et al. (eds.), *Protocols for High Speed Networks IV*

- Non-protocol-specific processing is a large part of the total load, as shown in [35]. Examples include interrupt handling, context switching and data copying at layer boundaries in deeply layered protocol stacks.
- The choice of hardware for the adaptor depends on the complexity of the functions it supports. In [2, 22] where the transport protocol layer is offloaded or in [7] where the full protocol stack can be offloaded, general purpose microprocessors are used. Probably because of the complexity of existing protocols, VLSI [24] implementation above the data link layer has been disappointing so far. In [8], dedicated VLSI chips are used to support TCP checksums. Also, some newer lightweight transport protocols are specially designed for VLSI implementation [1, 3].
- There is a tradeoff between performance, flexibility and cost. If the key functions of the frequently executed portion of the protocol remain relatively stable, there can be significant advantage in providing hardware support for these functions leaving the other tasks in the host software for flexibility.
- As host processing speed continues to outpace memory bandwidth and as the network bandwidth approaches the processor memory bandwidth, it is important to keep data movement on the workstations down to the minimum [4, 9, 28].

This paper presents a feasibility study for a new approach to hardware assistance. It combines the relatively simple operations needed for data transfer across multiple layers and provides a hardware "fast path" for them, which will be efficient for bulk data transfer. It is based on the "protocol bypass concept" [37] which is a generalization of Jacobson's "Header Prediction" algorithm [20] for TCP/IP. Bypass solves the problems identified above, which may limit the use of offboard processing, by implementing an entire service through all layers for certain cases. This simplifies the interface between the host and the adaptor chip and minimizes their interaction, which is supported by an access test, some DMA processing and a simple command protocol. The chip design based on bypassing is called ROPE, for Reduced Operation Protocol Engine. The contribution of this paper is to define the host/chip interface and the chip operation, and to report on a VHDL-based feasibility study of the chip design. It appears to be feasible to support an end-system single-connection data rate approaching 1 Gbps.

The next section introduces the bypass concept, its architecture and implementation. Section 3 analyzes the key protocol processing overheads and discusses the requirements for a bypass VLSI implementation. Sections 4, 5 and 6 describe a design study of a ROPE chip using the industry standard hardware description language, VHDL, with conclusions in Section 7.

2 The Bypass Concept

A bypass adds an additional path for certain operations, with minimal changes to the original software. Conformance to the protocol is maintained by doing all the other operations through the normal "heavyweight" path. A bypass path can be provided for send, for receive, or for both together, and is compatible with other end-systems implemented without a bypass.

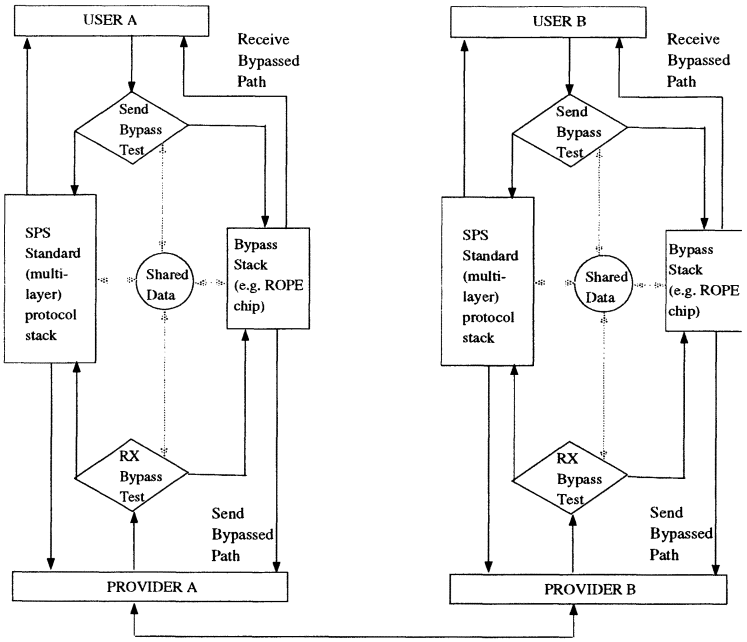

Figure 1 Bypass Architecture

2.1 Bypass Architecture

Figure 1 illustrates the architecture of a bypass implementation for any standard protocol. The standard protocol stack (SPS) is the processing path taken by all PDUs during a connection without the bypass. The SPS may refer to a single layer or to multiple adjoining layers of a layered protocol stack. The bypass has 4 key components:

- *Send Bypass* Test;
- *Receive Bypass* Test;
- *Bypass Stack*;
- *Shared data* for access by the two tests, the Bypass stack and the SPS.

The send bypass test identifies outgoing packets that are data packets in the data transfer phase. The receive bypass test matches the incoming PDU headers with a template that identifies the predicted bypassable headers. The bypass stack performs all the relevant protocol processing in the data transfer phase. The shared data are used to maintain state consistency between the SPS and the bypass stack, including window flow control parameters and connection identifiers. Whenever there is a change in the processing path between the SPS and the bypass stack, checks are performed to ensure that there are no outstanding packets in the current path, i.e. "no in-transit PDUs", before the change is made. A more detailed discussion on this is presented in another paper [33].

2.2 Efficient logic for the bypass test

The "no-in-transit PDU" test can often be avoided. At the beginning of data transfer on a new connection, it is automatically satisfied. It holds as long as no packet fails a bypass test, and it is sufficient to maintain a flag to indicate this. Once a packet fails, and goes to the SPS, then a full "no-in-transit PDU" test must be performed for each packet until the test succeeds, after which control can go back to the flag. Token management and synchronization points of the session layer [19, 18] which are mapped by equivalent application and presentation service primitives can be used as synchronization points within the bypass architecture, as it switches between the SPS and the bypass stack. Since they are only inserted periodically in bulk data transfer and can be controlled by the application process, these overheads are not excessive.

2.3 Multiple-layer bypass

A bypass for multiple layers instead of just one gives additional gains by avoiding:

- ☐ Overhead of encoding and decoding the interface control information passed between layers;
- ☐ Executing the full general protocol logic for the layers to decide how to manipulate the data;
- ☐ Queueing of data at layer boundaries.

The advantage is increased further in cases where some layers, like the network and application layers, have been further subdivided into sublayers.

A multiple-layer bypass path is a concatenation of processing procedures performed by the adjacent layers when they are simultaneously in the data transfer phase. Meanwhile, the separate layers in the SPS path handle the other phases.

In summary, the separation of the bypass path offers the following advantages:

- ☐ The processing path of data PDUs can be optimized;
- ☐ The number of possible PDU formats in the bypass path is reduced to data transfer PDUs;
- ☐ The finite state machine of the protocol is now reduced to only the "OPEN" state, for as long as processing remains in the bypass path. The state of the system does not change during the entire data transfer phase and the protocol processing is reduced to ensuring reliable transfer of data across the communications network.

3 Design Considerations for a Hardware Bypass

3.1 Factors affecting system performance

This section summarizes the major factors affecting throughput performance in a deeply layered stack on an end system. It follows the description by Heatley and Stokesberry [16].

Protocol procedures can be characterized as per-octet, per-packet or per-group-of-packets operations. The per-octet operations take an average time A per octet (e.g., checksum), and the per-packet procedures take time B per packet (e.g. address decoding and multiplexing). Per-group-of-packets operations include for example transmission of acknowledgments, whose frequency is implementation-dependent, and their timing will be aggregated and included in parameter B.

The throughput bound imposed by protocol processing alone, λ_{max} in octets per second, is then given by the equation:

$$\lambda_{max}(x) = \frac{x}{A.x + B.\lceil x/M \rceil} \tag{3.1.1}$$

where x is the size of the user message in octets and M is the maximum PDU size. In bulk data transfer, as x becomes large,

$$\lim_{x \to \infty} \lambda_{max}(x) = \frac{1}{A + B/M} \tag{3.1.2}$$

The protocol processing load on an end system is typically shared between the host and the network adaptor. As the raw data bit rate supported by optical networks approaches the main memory bandwidth of the end system, the cost of moving data and of per-octet processing limits the effective throughput presented to the application process, especially for bulk data transfer. The data portion of a PDU may be physically moved for the following reasons:

- Copying between the adaptor buffer and the host system memory;
- Crossing protection domains (address spaces) — e.g. at the user/kernel boundary. This problem becomes more pronounced in microkernels which treats a protocol task as a server process outside the kernel domain [11];
- Per-octet processing like presentation conversion and checksum routines.

Hardware implementation is particularly efficient for per-octet operations.

3.2 Requirements of a bypass VLSI implementation

The problems associated with separate or offboard processing, which were discussed in the introduction, are addressed by the VLSI design as follows:

- ☐ A clean separation of functionality requiring only a simple protocol to communicate between the host and adaptor is desired, and is provided by a bypass. Its particular set of functions are complete in themselves and have a focussed interface with the host software at the packet entry point. There is relatively infrequent switching between the SPS and the bypass stack;
- ☐ Reduced non-protocol-specific processing overhead. For example the processing of acknowledgment packets is dominated by interrupt handling, typically a few hundred instructions, rather than by the protocol processing itself. Our approach removes acknowledgment handling altogether from the host. Also, the bypass system can be extended to incorporate multiple-layer stacks and remove overhead that way;
- ☐ VLSI implementation complexity: only the data transfer functions are implemented.
- ☐ Tradeoff between performance, flexibility and cost. The host software processes non-data-transfer packets which are typically small but require more flexible and more complex processing. They are also only per-packet, so they have less impact overall. They can be efficiently handled by the host given the projected increase in host processing speed [17]. The operations implemented in VLSI are understood to have less need of flexibility.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.