Enhancing Integrated Layer Processing using Common Case Anticipation and Data Dependence Analysis
Extended Abstract

Philippe Oechslin
Computer Networking Lab
Swiss Federal Institute of Technology
DI-LTI EPFL
CH-1015 Lausanne, Switzerland
Telephone: +41 21 693 47 08
Fax: +41 21 693 66 00
Email: oechslin@ltisun.epfl.ch

Stefan Leue
Institute for Informatics
University of Berne
Länggassstrasse 51
CH-3012 Berne, Switzerland
Telephone: +41 31 631 44 30
Fax: +41 31 631 39 65
Email: leue@iam.unibe.ch

1 Introduction

Integrated Layer Processing (ILP) has been recognized as one efficient way to speed up the execution of protocols (for an introduction to ILP see [CT90]). Protocols, however, are still being specified in distinct layers and it is the job of the implementor to use the concepts of ILP with his intuition when designing an implementation of a protocol stack. In this paper we want to discuss under which conditions ILP can be applied to a given protocol stack and how an implementation can be structured to allow more ILP. Based on the methods discussed in this paper we intend to develop automated tools which lead from a formal specification of a protocol stack to a structured implementation allowing a maximum of efficiency gained by ILP. The methods have mainly been published in [LO93] and [LO94b].

1.1 Classical Protocol Stack Model

A protocol stack is usually specified as a set of concurrent processes each of which represents the functionality of one layer of the protocol stack. A packet is usually received at one end of the protocol stack and processed by each layer process in turn until it reaches the other end of the protocol stack. Since the different layer processes are concurrent several packets could be processed at the same time in a protocol stack, each one being in a different layer. This kind of processing is referred to as pipelining.

2 ILP and High Speed Protocols

Integrated layer processing consists in avoiding pipelining and processing each packet by its own. The processing steps in each layer are combined to form a unique thread of processing from where the packet enters the protocol stack to where it exits. Especially for data manipulation operations a combined execution of operations along the packet processing thread can be much more efficient than the separate executions of different processing steps. Moreover, depending on the type of implementation, the actual passing of packets from one layer to an adjacent layer can induce an overhead which is avoided when doing ILP.

We conjecture that high speed protocols can be defined as being those protocols in which the machine doing the processing is the bottleneck. With this definition TCP over Ethernet has been a high speed protocol at the time where it first appeared as are ATM based protocols nowadays. In both cases ILP is necessary to reduce the excessive load on the protocol processing machine.

2.1 Conditions for efficient ILP

A first condition for efficient ILP is that the processing of a packet through a protocol stack may not be blocking. If for example a packet can not be processed in the last layer of a protocol stack because an acknowledgement is pending, an integrated processing of all layers is not possible until the missing acknowledgment is received. In such a case it is more efficient to process all non-blocking layers while waiting for the acknowledgment and then to have only the last layer to process once the acknowledgment has been received.

The second condition for the possibility of efficient ILP is that the operations in the different layers can actually be combined into more efficient ones. For this to be true there may be no (control or data) dependences that enforce a sequential execution of operations. If for example a first operation in one layer depends upon a result which in turn depends on the execution of a second operation then the two operations must be executed sequentially and cannot be combined.

2.2 Enhancing ILP

From the first condition we see that protocols should be designed in such a way that there should be no blocking within the protocol stack. A blocking wait on an acknowledgement is typically avoided by using a sliding window mechanism. If a protocol is designed in such a way that the machine has to spend time waiting for some event to occur then it doesn't match our definition of high speed protocol and ILP is not the correct way of gaining efficiency.

From the second condition we see that dependences are a crucial point in integrated processing. Thus we propose a method which starts with a dependence analysis of the protocol stack and allows to detect and reduce the obstructive dependences. Our method is based on SDL specifications of protocol stacks for which it generates data and control flow dependence graphs. For details see [LO94a] and [LO94b].

3 Dependence graphs for protocol stacks

Each layer of a protocol stack in an SDL specification typically consists of a process defined as an extended finite state machine. For every possible transition of this machine a sequence of operations is specified. This sequence may include decisions and can thus be branching. There are two kinds of dependences between operations of a transition: data dependences and control dependences. Control dependences represent the sequence in which the operations are specified and the decisions that are made during execution. Data dependences represent the fact that the result of one operation has to be known before another operation can be executed correctly. For a definition of data and control dependences see [PKL80].

The dependences between the operations in a transition are best described using a dependence graph. The dependence graph for one transition (TDG) is found by a syntactical analysis of the statements executed in the course of one transition. One dependence graph is created for each entry point (point at which a packet is received from the environment) of the protocol stack. The graph is composed by concatenating TDGs of the transitions that can be triggered from the point where a packet is received at one end of the stack to the point where it is either destroyed or delivered at the other end. As decisions are taken during processing of the packet, this graph has the shape of a tree. The concatenation of the transitions of the different processes does not in general correspond to the SDL model of processes communicating through asynchronous queues. However, we argue that the concatenated execution of the transitions corresponds to one possible interleaving of the execution of the SDL model and that an implementation allowing just this one sequence of operations along the concatenated structure is thus valid.

4 Common/uncommon partitioning

The major part of a protocol usually serves to detect and recover from many kinds of exceptions and errors. However, in typical high speed environments (optical transmission) errors tend to be very rare. Thus a large part of a protocol is executed only in very few cases whereas the part executed most often is only a small fraction of the protocol's code. Which part of a protocol is common can be found either by simulation or by statistical analysis of a working model. Many assumptions can also be based on common sense reasoning (e.g., in a file transfer protocol the establishment of a connection is uncommon whereas the transfer of data is common). We partition our dependence graphs into common and uncommon parts by examining each decision and defining which evaluation of the decisions are common or uncommon. All nodes which depend transitively of some uncommon decision are marked uncommon and removed from the graph. The resulting graph is called the Common Path Graph CPG.

5 Avoiding decisions through anticipation

Decisions introduce 'hard' dependences similar to data dependences in the way that no next operation can be executed before the decision has been evaluated itself. Avoiding decisions thus reduces these dependences and enhances ILP. By anticipating the common case all decisions that have only one common outcome can be ignored. The consequence of this is that protocol processing can then be much more efficient for common cases. Another consequence is that inconsistencies appear in the uncommon cases. To treat these inconsistencies some precautions have to be taken.

6 Relaxing the dependences

The basic idea is that we want to eliminate control flow dependences which impose sequential processing of operations as far as possible. Thus we generate a relaxed dependence graph which consists only of the data flow dependences of the complete dependence graph. In order to ensure that the resulting graph structure is connected and that all decisions are respected we have to introduce some auxiliary dependences.

7 Grouping Data Manipulation Operations (DMOs)

The joint execution of two or more operations on the same data has been proven to greatly enhance the efficiency of a protocol as the total number of data relocation operations can be reduced. We propose an algorithm which is a generalization of the methods proposed in [AP93] and [OP91]. It has the advantage of combining DMOs at compile time, taking into account all combinations that may be needed at run time.

8 A small example

In the full paper we give a small example with an SDL specification and all the dependence graphs.

9 Conclusion

After having discussed some limiting factors for ILP we have explained a formal method which allows to derive an optimized ILP protocol implementation from the formal SDL specification of a protocol. Starting with a dependence analysis we create a common path graph for each entry point of the protocol stack. This graph represents the processing that has to be carried out in the common case in all protocol layers until the packet can be delivered to the environment. Considering only the common case not only reduces the complexity of this graph but also allows to anticipate the outcome of some decisions and consequently to reduce control dependences. The CPG is transformed to reflect only the data dependences and some control dependences that have to be respected in the common case which increases the potential parallelism. The transformed dependence graph is then the starting point for an algorithm that allows to generate, at compile time, all the combinations of DMOs necessary for the execution of the common cases. We believe that our method is a contribution towards the integration of ILP concepts into automatic implementation methods based on formal specifications.

References

[AP93] M. Abbot and L. Peterson. Increasing network throughput by integerating protocol layers. *IEEE/ACM Transactions on Networking*, 1(5):600–610, october 1993.

[CT90] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM SIGCOMM '90 conference*, Computer Communication Review, pages 200–208, 1990.

[LO93] Stefan Leue and Philippe Oechslin. Optimization techniques for parallel protocol implementation. In *Future Trends of Distributed Computing Systems*, pages 387–393. IEEE, 1993.

[LO94a] S. Leue and Ph. Oechslin. A formal approach to optimized parallel protocol implementation. Technical report, University of Berne, Institute for Informatics, 1994.

[LO94b] S. Leue and Ph. Oechslin. Formalizations and algorithms for optimized parallel protocol implementation. In *International Conference on Network Protocols*, 1994. to appear.

[OP91] S. W. O'Malley and L. L. Peterson. A highly layered architecture for high-speed networks. In M. J. Johnson, editor, *Protocols for High Speed Networks II*, pages 141–156. Elsevier Science Publishers (North-Holland), 1991.

[PKL80] D. Padua, D. Kuck, and D. Lawrie. High-speed multiprocessors and compilation techniques. *IEEE Transactions on Computers*, 29(9):763–776, September 1980.