

# Reducing Protocol Ordering Constraints to Improve Performance

David C. Feldmeier and Anthony J. McAuley

Computer Communication Research Group, Bellcore,  
445 South St., Morristown, NJ 07962; dcf, mcauley@bellcore.com

## Abstract

Multipath routing and data retransmission cause data misordering. Although data could be reordered before processing, it is simpler and more efficient to process data as it arrives. Because existing protocol functions generally cannot process misordered data, we need new functions with minimal ordering constraints. For two example functions, CRC error detection and CBC mode encryption, we show that the functions have ordering constraints and we present new functions that provide similar functionality without ordering constraints.

Keyword Codes: C.2.2

Keywords: Network Protocols

## 1 Introduction

While developing a high-speed protocol processing host-network interface for the AURORA gigabit network, we came across an interesting problem associated with packet misordering. Some protocol functions cannot be computed on misordered data, and thus, the data must be reordered before processing. We demonstrate the problem using cyclic redundancy code (CRC) error detection and cipher block chaining (CBC) encryption as examples. To solve the problem, we introduce new error detection and encryption functions.

AURORA is one of five gigabit testbeds sponsored by NSF and DARPA to explore technologies for the National Research and Education Network (NREN) [1]. The main objective of AURORA is the exploration of technologies and architectures appropriate for gigabit networks. The experiment discussed in this paper is shown in Figure 1. All of the lines carry 53-byte ATM packets with 48-byte payloads. The local network runs at 1.2 gbps ( $= 8 \times 155$  Mbps) and each line of the wide-area switched network runs at 155 Mbps (SONET OC-3). As packets arrive at the wide-

οργάνωση της εταιρείας  
για να πετύχουμε, φέρει στο φως





arbitrarily-ordered data.

We propose a solution in the form of new protocol processing functions that allow the processing of misordered data. Section 2 presents two example protocol functions that have ordering constraints: the CRC error detection code and the CBC encryption mode. Section 3 describes two specific protocol functions without ordering constraints to replace CRC and CBC encryption. In the Appendix we examine the general types of protocol functions that have ordering constraints and derive general methods of minimizing these constraints.

## 2 Example Functions with Ordering Constraints

In this section, we discuss two functions with ordering constraints that are used in communication protocols: the cyclic redundancy code (CRC) for error detection and cipher block chaining (CBC) for encryption and decryption.

### 2.1 Cyclic Redundancy Code

An error detection encoder takes  $k$  data bits and appends  $h$  bits based on the error detection code. The receiver takes the  $k + h$  bit PDU and checks to see if it is a valid codeword. If it is not a valid codeword, the whole PDU is rejected. Two key error detection parameters are the minimum distance between valid codewords ( $d$ ) and maximum error burst length that is always detected ( $b$ ).

A robust form of error detection is the CRC that is used in protocols such as HDLC [10]. The CRC has powerful error detection properties, with  $d = 4$  and  $b = h$ . Each CRC PDU can be processed independently; however, within a PDU data must be processed serially. The CRC treats each  $k + h$  bit PDU as the coefficients of  $C(x)$ , a polynomial of degree  $k + h$ . The values of the  $h$  bits are chosen such that the CRC generator  $G(x)$ , a polynomial of degree  $h$ , exactly divides  $C(x)$ :

$$C(x) = 0 \pmod{G(x)}.$$

At the receiver, the received data are treated as the coefficients of  $C'(x)$ . If  $C'(x) = 0 \pmod{G(x)}$ , then the received data is accepted as correct; otherwise it is discarded.

Polynomial division is the key operation for CRC error detection at the receiver. Because division uses conditional subtraction, the subtraction operations cannot be performed in an arbitrary order; each subtraction depends on the results of all previous subtractions. This feedback can be seen more clearly in the serial CRC implementation shown in Figure 3. It is not possible to process in arbitrary order with this structure; if one bit of the PDU is delayed, no other bits can be processed until that bit is received.

Because CRC is a linear code, it is true that  $CRC(x \oplus y) = CRC(x) \oplus CRC(y)$ , and thus the CRC of a block of data can be generated from the CRC of the individual pieces of data. Unfortunately, the amount of computation necessary for the CRC





Two general ways in which functions can be applied to atoms are:

1.  $f_s(\text{state}, \text{atom}) \rightarrow \text{state}'$
2.  $f_a(\text{state}, \text{atom}) \rightarrow \text{atom}'$

where an atom is data contained in a PDU and state is kept by the protocol processor per PDU. The first type of function changes the PDU state, but does not change the atom ( $f_s$ ). The second type of function changes the atom, but does not change the PDU state ( $f_a$ ). Protocol operations can be either of these functions or a combination of both. The CRC error detection code is a  $f_s$  function. The CBC encryption is a combined  $f_a/f_s$  function.

For error detection and encryption, specific functions without ordering constraints are described in the next section. The general concepts for designing functions without ordering constraints are described in the Appendix.

### 3 Example Solutions

This section describes new functions without ordering constraints that can be used for protocols instead of CRC error detection and CBC encryption.

#### 3.1 Error Detection

Error detection is an example of an  $f_s$  function. The CRC has desirable error detection properties ( $d = 4$  and  $b = h$ ) but imposes ordering constraints. A Fletcher checksum also imposes ordering constraints. The TCP checksum does not have ordering constraints, since summation is commutative. Unfortunately, the TCP checksum has poor error detection properties ( $d = 2$  and  $b = h - 1$ ). This means, for example, that 2-bit errors could produce an undetected error in the TCP checksum, while the CRC will detect all 2-bit and 3-bit bit errors. We have designed a code that has desirable error detection properties ( $d = 4$  and  $b = h$ ) and imposes no ordering constraints. We call this code WSC-2 [7].

Like the CRC, a WSC-2 encoder takes  $k$  bits of data and appends  $h$  bits for parity, treating this group of  $k + h$  bits as the coefficients of  $C(x)$  a polynomial of degree  $k + h$ . The WSC-2, however, blocks  $C(x)$  into  $\frac{2(k+h)}{h}$   $\frac{h}{2}$ -bit polynomials ( $R_i(x)$ ) of degree  $\frac{h}{2}$ , where the final two polynomials ( $R_1(x)$  and  $R_0(x)$ ) represent the two  $\frac{h}{2}$ -bit parity atoms. The values of two parity atoms are chosen such that:

$$\sum_{i=0}^{\frac{2k}{h}+1} R_i(x) = 0$$

$$\sum_{i=0}^{\frac{2k}{h}+1} i \cdot R_i(x) = 0 \pmod{M(x)}$$





as it arrives. Existing protocol functions generally cannot process misordered data. We derive some simple guidelines for choosing functions with minimal ordering constraints because misordering cannot always be prevented (see the Appendix). For error detection and encryption, we show that commonly-used functions have ordering constraints and we present new functions that provide similar functionality without ordering constraints. Also, McAuley has designed a forward error correction code without ordering constraints[8].

In addition to the other advantages of functions without ordering constraints is the simplification of host-network interface design. Consider the processing of a TCP segment that arrives in a single packet. The TCP checksum must be computed over most of the packet, but where the checksumming begins depends on the TCP header. Performing the checksumming operation on the network interface is easy and fast, but finding the point at which to begin checksumming is complex. A compromise is to checksum the entire packet using network interface hardware, and then perform a checksum in software over the parts of the packet that should not have been checksummed. The difference of these two checksum values is the correct TCP checksum. This hybrid checksumming technique reduces the amount of checksumming that must be done in software and yet the network interface remains simple<sup>1</sup>. Notice that the hybrid checksumming technique can be generalized to work with any function that has no ordering constraints and easily computed inverses, such as the WSC error detection code described in this paper.

As far as we know, we are the first people to deal with non-reordered processing of PDU segments on a misordering network. Others have dealt with two of the three problems. The Datakit URP protocol [6] deals with non-reassembled processing of PDU segments, but the Datakit network does not misorder. TCP fragmentation deals with PDU segments on a misordering network, but requires reassembly before processing. Non-reordered processing of entire PDUs is straight-forward.

We believe the removal of ordering constraints is important for high speed communication protocols. Although the algorithms described in this paper were designed for TP++, the ideas are generally applicable to other protocols. In fact, our collaborators in the AURORA project at the University of Pennsylvania are planning to use the encryption mode described in this paper to allow pipelining in their encryption hardware [3].

## Acknowledgments

Our thanks to Chase Cotton for his extensive discussions of the ideas in this paper. Our thanks also to Al Broscius, Ernst Biersack, Bruce Davie, Paul Lin and Dave Sincoskie for their helpful comments.

---

<sup>1</sup>The hybrid hardware/software checksumming of a TCP segment has been implemented in a network interface built by Greg Watson, Dave Banks and Mike Prudence of HP Labs, Bristol[11].

## References

- [1] E. W. Biersack, C. J. Cotton, D. C. Feldmeier, A. J. McAuley, and W. D. Sincoskie, “Gigabit Networking Research at Bellcore”, *IEEE Network*, 6(3):42–48, March 1992.
- [2] E. W. Biersack, “Efficient Connection Management Using Synchronized Clocks”, In *Proc. IFIP Third Workshop on High Speed Networking*, Berlin, Germany, March 1991.
- [3] A. Broscius and J. M. Smith, “Exploiting Parallelism in Hardware Implementation of the DES”, In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pp. 367–376, Springer-Verlag, 1992.
- [4] D. D. Clark and D. L. Tennenhouse, “Architectural Considerations for a New Generation of Protocols”, In *Proc. ACM SIGCOMM '90*, pp. 200–8, Philadelphia, PA, September 1990.
- [5] D. E. Denning, *Cryptography and Data Security*, Addison Wesley, 1983.
- [6] A. G. Fraser and W. T. Marshall, “Data Transport in a Byte Stream Network”, *IEEE Journal on Selected Areas in Communications*, 7(7):1020–1033, September 1989.
- [7] A. J. McAuley, “Weighted Sum Codes for Error Detection and Their Comparison with Existing Codes”, available via anonymous FTP on thumper.bellcore.com in pub/tp++/error\_detection\_new.
- [8] A. J. McAuley, “Reliable Broadband Communication Using a Burst Erasure Correcting Code”, In *Proc. ACM SIGCOMM '90*, pp. 297–306, Philadelphia, PA, September 1990.
- [9] B. Pehrson, P. Gunningberg, and S. Pink, “Distributed Multimedia Applications on Gigabit Networks”, *IEEE Network Magazine*, 6(1):26–35, January 1992.
- [10] A. S. Tanenbaum, *Computer Networks*, Prentice-Hall, Inc., second edition, 1988.
- [11] G. Watson, personal communication.

## A Appendix - Protocol Ordering Constraints

Protocol functions can constrain the acceptable processing orders of received data. This section discusses the general types of processing functions and how protocol functions should be chosen to minimize ordering constraints.

## A.1 Protocol Model

We assume that protocol processing requirements are defined in the protocol specification. Examples of these functions include error detection (e.g. CRC) and data format conversion (e.g. encryption). Protocol functions are performed on the payload of PDUs, and the PDU headers contain control information that influences the function. Thus all data of a PDU are processed identically but data in different PDUs may be processed differently.

Ordering constraints can be placed into two categories: inter-PDU constraints and intra-PDU constraints. PDUs can be processed independently because each carries its own control information; thus there are no inter-PDU ordering constraints. Because inter-PDU constraints need not exist, we will concentrate on intra-PDU ordering constraints.

Before we derive intra-PDU ordering constraints, we must discuss what elements exist within a PDU that can be misordered. Regardless of the protocol function that operates over a PDU, there are minimum units over which processing occurs. Because these units cannot be usefully subdivided, we refer to these as the *atoms* of the function. The atoms are usually inherent to the algorithms. For example, the DES encryption algorithm has an atom of 64 bits.

Two general ways in which functions can be applied to atoms are:

1.  $f_s(\text{state}, \text{atom}) \rightarrow \text{state}'$
2.  $f_a(\text{state}, \text{atom}) \rightarrow \text{atom}'$

where an atom is data contained in a PDU and state is kept by the protocol processor per PDU. We will refer to this state as the PDU state for short. The first type of function changes the PDU state, but does not change the atom ( $f_s$ ). The second type of function changes the atom, but does not change the PDU state ( $f_a$ ). Protocol operations can be either of these functions or a combination of both.

CRC is an  $f_s$  function that takes the current PDU state (the partially-computed CRC) and an atom of data to produce a new PDU state. The atom of data is unchanged. Block encryption is an  $f_a$  function that uses the PDU state (derived from the key) and an atom of data to produce an atom of encrypted data. The PDU state remains unchanged. Cipher block chaining uses both  $f_s$  and  $f_a$  functions; the current PDU state (derived from the key and result of the previous  $f_s$  operation) and an atom of data to produce an atom of encrypted data and a new PDU state. Both  $f_s$  and  $f_a$  start with the same PDU state and operate in parallel.

Our protocol model is not complete; it only models the aspects of the protocol that are concerned with data processing. For example, we discuss processes such as error detection, but we do not discuss other aspects of error control, such as acknowledgments. Although additional functionality could be added to the model, it is beyond the scope of this paper.

## A.2 Derivation of Ordering Constraints

From the  $f_s$  and  $f_a$  functions, we can derive intra-PDU ordering constraints. Both types of functions require an initial state. If the initial state is not defined in the protocol specification, then the initial state must be provided to the function before any data processing can occur. Normally this initial state is contained in the PDU header; thus if initial state is needed, then the PDU header must arrive before any PDU data can be processed. This ordering constraint cannot be eliminated.

### A.2.1 $f_a$ Functions

For  $f_a$  functions, because the state does not change as data is processed, the only ordering constraint that applies is that the initial state must arrive before any associated data is processed.  $f_a$  functions have the weakest ordering constraints, and no changes can be made to reduce further the ordering constraints.

### A.2.2 $f_s$ Functions

Now let us consider  $f_s$  functions. Part of the PDU processing may be to check the final state of the system against a reference state and perform an appropriate action (e.g. accept or reject the data as correct). There are two ordering considerations. One is when the reference state arrives relative to the atoms. Because the final state of the PDU is not known until all atoms have been processed, it does not matter when the reference state arrives relative to the data. The second ordering consideration is the relative processing order of the atoms.

For  $f_s$  functions, only the final state after a PDU has been processed completely is important. The intermediate states during processing are of no consequence. Therefore, all processing orders of atoms that lead to the same final state as serial processing of atoms are acceptable. The larger the number of acceptable processing orders, the fewer the ordering constraints, and the more efficient the implementation. For an  $f_s$  function to perform correctly regardless of the arrival order of data requires that the  $f_s$  function be both commutative and associative. If the function is not associative, then processing must be done in the original transmission order. If the function is associative, but not commutative, then an atom can be processed if it is adjacent to atoms that already have been processed. This is really not much better than processing in the original transmission order. Our goal should be to choose functions that are both commutative and associative.

### A.2.3 Combined $f_a$ and $f_s$ Functions

Some PDU processing may combine  $f_s$  and  $f_a$  functions:

1.  $f_s(\text{state}, \text{atom}) \rightarrow \text{state}'$
2.  $f_a(\text{state}, \text{atom}) \rightarrow \text{atom}'$

By combined functions, we mean that the result of the function is as if both the  $f_a$  and the  $f_s$  functions were executed simultaneously with identical states and atoms. For combinations of  $f_s$  and  $f_a$  functions, both the final state of the  $f_a$  function and the atoms output by the  $f_s$  function must be the same as if we did the atom processing in their original transmission order. Because the  $f_a$  function takes state as a parameter, for the output atom to be correct, the state must be the same as if serial atom processing had been done by  $f_s$ . With this inter-relationship, the only way to guarantee that the state is the same as that produced by serial atom processing is to process the atoms in serial order. Therefore, no improvement in ordering constraints is possible for combined  $f_s$  and  $f_a$  functions. The only way to fix this is to eliminate the link between the  $f_s$  and  $f_a$  functions.

We can eliminate the link between the  $f_s$  and  $f_a$  functions by noticing that the  $f_s$  function performs two duties: it updates state and it causes the  $f_a$  function to process atoms differently based on their position within the PDU. Thus we can introduce a new  $f_a$  function that depends not only on the initial state of the system and the received atom, but also the atom's position within the PDU. The new pair of functions is:

1.  $f_s(\text{state}, \text{atom}) \rightarrow \text{state}'$
2.  $f_a(\text{initial\_state}, \text{atom}, \text{atom\_sequence\_number}) \rightarrow \text{atom}'$

The new  $f_a$  is independent of  $f_s$  and has minimal ordering constraints. If the  $f_s$  function is chosen to be both commutative and associative, then the new pair of functions has minimal ordering constraints.