8

INTEROPERABILITY TEST SUITE GENERATION FOR THE TCP DATA PART USING EXPERIMENTAL DESIGN TECHNIQUES

Jiwon Ryu[†], Myungchul Kim[†], Sungwon Kang[††], and Soonuk Seol[†]
† Information and Communications University
{jwryu, mckim, suseol}@icu.ac.kr
†† Korea Telecom Research & Development Group
kangsw@sava.kotel.co.kr

Abstract Test derivation methods suitable for interoperability testing of communication protocols were proposed in the literature and applied to the TCP and the ATM protocols. The test cases that were generated by them deal with only the control part of the protocols. However, in real protocol testing, the test cases must manage the data part of them as well. For complete testing, in principle all possible values of the data part must be tested although it is impractical to do so. In this paper, a method is presented for generating the interoperability test suite for both the data part and the control part with the example of TCP connection establishment. In this process, experimental design techniques from industrial engineering are used to reduce the size of test suite while keeping a well-defined level of test coverage. Experimental design techniques have been used for protocol conformance testing but not for interoperability testing so far. We generate the test suite for the TCP data part by this method and show a possibility that we can test interoperability of protocols with the reduced number of test cases with a well-defined level of test coverage.

Keywords: protocol testing, interoperability testing, test suite generation, control part, data part, TCP, experimental design techniques, orthogonal arrays, factor, level

1. INTRODUCTION

With the wide spread of the telecommunication equipment and services, it is considered essential to guarantee interoperability of communication protocol implementations residing in them. As an attempt to guaranteeing

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35516-0_20 interoperability, conformance testing methodology has been published as international standards in ITU-T X.290 Series [6] and ISO/IEC 9646 [5], and used world-wide. Protocol conformance testing is used for confirming whether or not the behavior of Implementation Under Test (IUT) conforms to its standards and specifications, and promotes the probability of interoperation among IUTs. However, even though equipment and services successfully passed conformance testing, they do not often interoperate owing to the variety of mandatory parameters, optional parameters and control scopes. For this reason, interoperability testing is needed to check the interaction behavior among IUTs implemented according to their specifications. So far, no accepted common methodology for interoperability testing has existed.

Paper [8] proposed an algorithm for generating the interoperability test suite for communication protocols, and [13] and [14] respectively applied it to the Internet TCP protocol and the ATM/B-ISDN signaling protocol. The derived test cases are based on Finite State Machine (FSM) and the test generation methods consider only the control part of the protocols. So far, there has been no work on generating the interoperability test suite considering the data part of protocols.

Because most protocols have a variety of data variables, for real protocol testing the test suite generation has to deal with the data part as well as the control part of them. The size of test suite generated with the test derivation method considering the data part of protocols becomes much larger than that considering only the control part of them. In order to reduce the size of test suite while maintaining a well-defined level of test coverage, we make use of experimental design techniques from industrial engineering. They are used for planning experiments so that one can extract the maximum possible information from as few experiments as possible. Adopting these techniques leads to faster detection of non-interoperation of protocols while maintaining a well-defined level of test coverage. Experimental design techniques have been used for protocol conformance testing but not for interoperability testing so far. In this paper, we present an interoperability test suite derivation method considering the data part of protocols. It is illustrated with the example of TCP connection establishment. We then apply experimental design techniques to the test suite and compare the size of the test suite generated by the techniques with the size of test suite generated without the techniques.

This paper is organized as follows: in Section 2, as background we describe the interoperability test generation method for the control part of the TCP in paper [13], experimental design techniques, and the specifications of TCP connection establishment. In Section 3, with the experimental design techniques under certain assumptions, we generate the interoperability test suite for the data part of TCP connection establishment. Finally in Section 4, we discuss the conclusion and the future work.

2. RELATED WORK

In this section, we survey the interoperability test suite generation method for the TCP control part, explain experimental design techniques to be used to reduce the size of the interoperability test suite for the TCP data part, and describe the specification of TCP connection establishment.

2.1 Interoperability Test Suite Generation for the TCP Control Part

Paper [13] implemented the algorithm for testing interoperability for the class of communication protocols proposed in paper [8] and applied it to the TCP. The implemented program generates the interoperability test suite after an FSM is given. Figure 1 represents the connection establishment phase of the TCP FSM shown in paper [13]. Input is given by the application program running on TCP and output is presented by selecting one or more bits among URG, ACK, PSH, RST, SYN, and FIN in the 6-bit control field of TCP packet. TCP controls the connection establishment by selecting these bits.

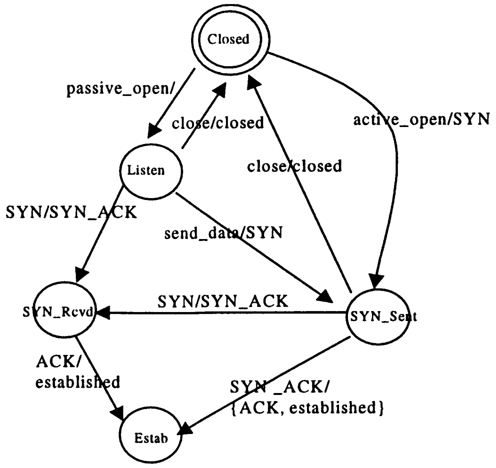

Figure 1. Connection establishment phase of TCP FSM.

As the result of executing the program of which input is TCP FSM in Figure 1, twelve interoperability test cases shown in Appendix 1 are derived. The following test case is for a three-way handshake and equivalent to item <1> of Appendix 1:

(Closed,Listen) -- active_open_a/[<,i_SYN,,i_SYN_ACK>, <established,i_ACK,established,>] → (Estab,Estab)

(Closed,Listen) is the starting stable state[1], 'active_open_a' before '/' is an input symbol to IUT A, and '[<,i_SYN,,i_SYN_ACK>, <established,ACK,established,>]' after '/' is a set of output symbols. Corresponding to the input symbol with postfix 'a' (or 'b') represents a message transmitted by Tester A (or Tester B) via interface A (or interface B) respectively. 'i_' is used to indicate internal messages as opposed to external messages. As shown in Figure 2, outputs are represented as a vector <u1, u2, u3, u4>, where u1, u2, u3, and u4 respectively represent messages transmitted by IUT A via interface A, IUT A via interface C, IUT B via interface B, and IUT B via interface C respectively. '→' is the transition relation and (Estab,Estab) is the arrival stable state. The test cases like this cannot be executed in real testing because they do not have values for the data part.

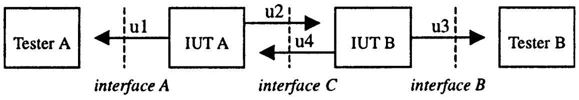

Figure 2. Output symbols of two FSMs.

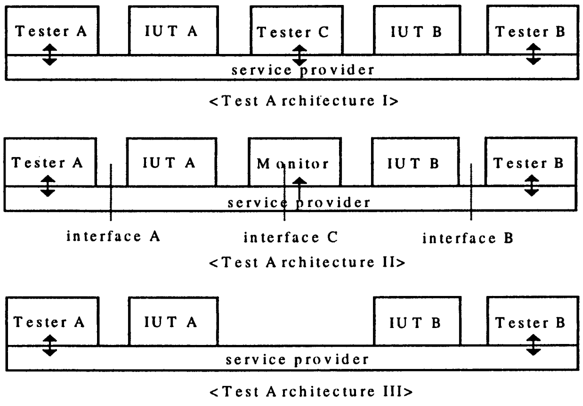

Figure 3. Interoperability test architectures.

Paper [14] presents three test architectures for the interoperability testing as shown in Figure 3. In Test Architecture I, Tester A sends a message to IUT A and then IUT A can send messages to IUT B and/or Tester A. If IUT B receives the message, it can also send messages to IUT A and/or Tester B. Tester C located between IUT A and IUT B can observe/modify messages

[1] A stable state is defined in paper [1] as a system state that is reachable from the initial state adhering to the single stimulus principle, and from which no change can occur without another stimulus.

between IUT A and IUT B. In Test Architecture II, however, Monitor located between IUT A and IUT B can only observe messages between IUT A and IUT B. In Test Architecture III, it is impossible to read these messages owing to the absence of Tester C or Monitor. Architecture III was considered in generating the test suite of the control part in paper [13]. In order to generate the test suite including data part, we have to observe values of internal messages between IUT A and IUT B. Therefore Architecture II is considered in this paper.

2.2 Experimental Design Techniques

Experimental design is a statistical technique for planning experiments and for choosing and analyzing data so that one can extract the maximum information from as few experiments as possible [9].

After deciding on the purpose of the experiment, we must choose factors and levels of the factors. The factors are defined as the various parameters of interest and the levels are defined as the values taken for each parameter. For example, if the temperature is chosen as a factor, 150°C or 200°C can be a level of the factor. The number of the different levels of the factor is defined as the space of the level.

When too many test cases are derived, we need to reduce the number of test cases while achieving the purpose of the experiment. In order to make the number of the test cases as small as possible, an assumption is needed such that the interactions of three or more factors virtually do not exist in the experiment. The assumption is that the risk of an interaction among three or more fields is balanced against the ability to complete system testing within a reasonable budget. An analysis of field data at Bellcore indicated that most field faults are caused by interactions of one or two fields [3, 4]. In the protocol world, it is also felt that most problems are caused by the interactions of a few state variables [3]. Thus the assumption for the experimental design techniques is satisfied in protocol testing area. This paper investigates this approach, using the method of orthogonal arrays to determine the test suite that cover all two-way interactions. Orthogonal arrays are test sets such that, for any pair of factors, all combinations of levels occur. The test suite using orthogonal array has more well-defined level of test coverage than other test suites with the same size [9, 10].

At Bellcore, the Automatic Efficient Test Generator (AETG) [2, 3, 4] was developed based on ideas of statistical experimental design theory to reduce the number of tests. The AETG is a system that generates test suite from user defined test requirements. AETG was used in Bellcore for screen testing and protocol conformance testing such as ISDN protocol conformance testing: call rejection and channel negotiation [1].

Paper [15] presented a guide to the theory and practical application of the method of orthogonal Latin squares to generate system test configurations that achieve pairwise parameter coverage.

2.2.1 Orthogonal Arrays

Orthogonal array design is a method requiring as few experiments as possible in an experiment with many factors [10]. Orthogonal array designs are test sets such that, for any pair of factors, all combinations of levels occur and every pair occurs the same number of times. So orthogonal array designs produce a test set of a manageable size that still covers the interactions that cause most of the field faults.

Orthogonal arrays are available with a variety of levels from 2 to 5. Depending on the levels, a method for making orthogonal arrays is different. For the majority of purposes, orthogonal arrays consisting of two or three levels should be sufficient. In this paper, two-level orthogonal arrays are referred. 2^m is the number of test cases and 2^m-1 is not only the number of columns but also the maximum factor to enable to arrange. The variable, m, is an integer of 2 or above. Thus two-level orthogonal arrays for 2^m-1 factors as the maximum enable to test out with 2^m test cases while covering interactions that causes most field faults.

To elaborate on these designs, consider a situation where three factors have two levels per a factor, say 1 and 2. In this case, the exhaustive test set has eight test cases, namely, (1,1,1), (1,1,2), (1,2,1), (2,1,1), (1,2,2), (2,1,2), (2,2,1), and (2,2,2). These test cases cover the interactions of three factors. A corresponding orthogonal array has four test cases, namely, (1,1,1), (1,2,2), (2,1,2), and (2,2,1). These test cases cover pairwise interactions.

Table 1. Orthogonal array for 3 factors, 2 levels.

Experimental Number	Column number (factor)		
	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Table 2. Number of test cases and breadth of coverage.

Experimental example	Number of test cases needed		Breadth of coverage (%)	
	Exhaustive testing	Orthogonal array	Exhaustive testing	Orthogonal array
3 factors, 2 levels testing	8	4	100	100

Table 1 represents the orthogonal array to enable to test three factors as the maximum with four test cases, as substituted 2 for m. In orthogonal arrays, the arrangement of factors and levels are randomly chosen. In this example, compared to the exhaustive testing, there is a 50% reduction in the number of test cases. It is possible to test with only four test cases because in the real world most problems are caused by the interactions of two factors as illustrated in section 2.2. As shown in Table 1, two columns chosen randomly include pairwise interactions: (1,1), (1,2), (2,1), and (2,2), so orthogonal array designs include all pairwise combinations of the test factors. Table 2 summarizes the number of test cases needed and the breadth of coverage for the exhaustive testing and orthogonal array designs. The breadth of coverage is defined here as the percentage of all pairwise combinations of the test factors.

2.3 TCP Specification for Connection Establishment

TCP consists of three phases, i.e., connection establishment, connection release, and data transmission. In this paper, since we generate the test suite for TCP connection establishment phase, we only give the specification of connection establishment phase. Figure 4 shows the format of the TCP header [12].

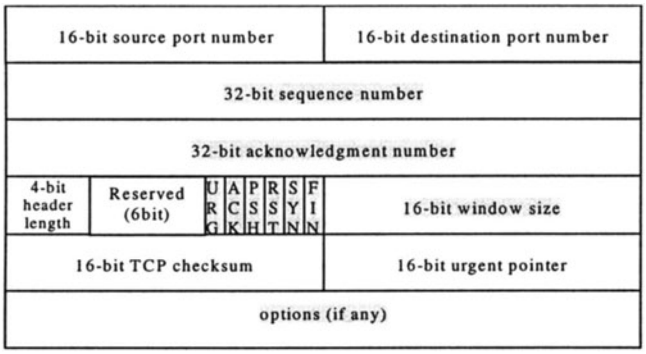

Figure 4. TCP header.

Since TCP is a connection-oriented protocol, a connection must be established between two ends before either end can send data to the other. TCP establishes connections with a procedure known as a three-way handshake. The TCP packets for this procedure include sequence number (seq), acknowledgement number (ack), window size (win), and maximum segment size (mss) as well as the control field. These fields are presented by the shadow in Figure 4 and are the factors related to test suite generation for the TCP data part.

Since the sequence number and the acknowledgement number are 32-bit fields, their space ranges from 0 to 2^{32}-1. Since the space is finite, they cycle from 2^{32}-1 to 0 again. When each end sends its SYN to establish a connection, it chooses an initial sequence number (ISN) for the connection. The ISN should change over time so that each connection has a different ISN. The ISN should be viewed as a 32-bit counter that increments by one every 4 microseconds because the sequence numbers on the clock are increased about every 4 microseconds [11].

TCP's flow control begins by each end advertising a 16-bit window size. Since the window size is significant only when combined with an acknowledgement number, this field is meaningful only when the acknowledgement field is valid. When the ACK bit in the control field is set, the requesting end sends a SYN segment with the window size. Some applications change their buffer sizes to increase performance, but the window size need not change its default because any data is not exchanged for the connection establishment phase.

The maximum segment size for option field is a 16-bit field and TCP uses this option only during connection setup. The sender advertises the maximum segment size and does not want to receive TCP segments larger than this value. This is normally to avoid fragmentation. For Ethernet this implies the maximum segment size of up to 1460 bytes.

Figure 5 shows the tcpdump [7] output for the segments for TCP connection establishment.

```
1 svr4.1037 > bsdi.discard: S 1415531521 : 1415531521(0) win 4096 <mss 1024>
2 bsdi.discard > svr4.1037: S 1823083521 : 1823083521(0) ack 1415531522 win 4096
  <mss 1024>
3 svr4.1037 > bsdi.discard: . ack 1823083521 win 4096
```

Figure 5. tcpdump output for TCP connection establishment.

These three TCP segments contain only TCP headers. No data is exchanged. For TCP segments, each output line begins with 'source > destination: flags' where flags represent the control bits. '1415531521:1415531521(0)' means the sequence number of the packet is 1415531521 and the number of data bytes in the segment is 0. In line 2 the field 'ack 1415531522' shows the acknowledgement number. This is printed only if the ACK flag in the header is on. The field 'win 4096' in every line of output shows the window size being advertised by the sender. The final field '<mss 1024>' in the lines 1 and 2 shows the maximum segment size option specified by the sender.

3. INTEROPERABILITY TEST SUITE GENERATION FOR THE TCP DATA PART

In this section, we describe a method to derive the test suite for the TCP data part from the test suite previously generated in paper [13]. Figure 6 shows stages for deriving the interoperability test suite for the TCP data part. In paper [13], the test suite for the control part was generated as the result of giving TCP FSM to the implemented program. The generated test suite for the control part has the effect to exclude impossible behavior sequences. In this work, the test suite for the data part is based on that for control part. In Section 3.1, we lay down some assumptions for test suite generation for the data part. In Section 3.2, based on the assumptions of Section 3.1 we generate test suite for the data part. In Section 3.3, by using orthogonal arrays, we again generate test suite from the test suite derived in Section 3.2. In Section 3.4, we calculate and compare the sizes of the test suites generated at various stages.

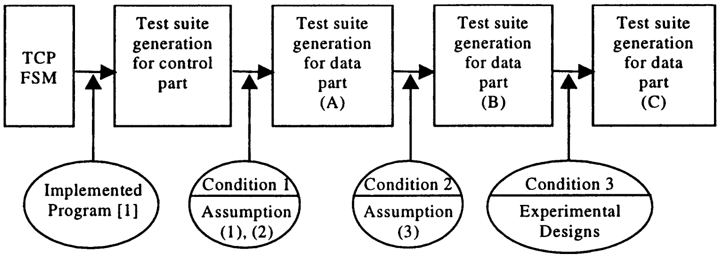
TCP FSM
Test suite generation for control part
Test suite generation for data part (A)
Test suite generation for data part (B)
Test suite generation for data part (C)
Implemented Program [1]
Condition 1
Assumption (1), (2)
Condition 2
Assumption (3)
Condition 3
Experimental Designs

Figure 6. Stages for generating the test suite for the data part.

3.1 Assumptions

We need assumptions for the test purpose to generate test suite for the data part based on the test suite for the control part.

(1) We consider six factors: the sequence numbers, the window sizes, and the maximum segment sizes of two TCPs.

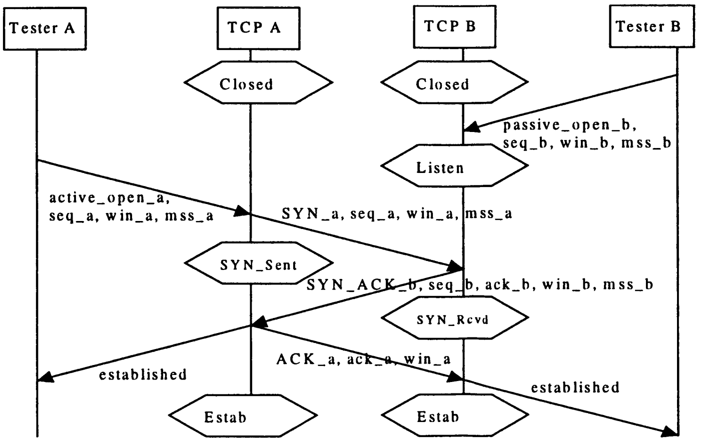
Tester A
TCP A
TCP B
Tester B
Closed
Closed
passive_open_b, seq_b, win_b, mss_b
Listen
active_open_a, seq_a, win_a, mss_a
SYN_a, seq_a, win_a, mss_a
SYN_Sent
SYN_ACK_b, seq_b, ack_b, win_b, mss_b
SYN_Rcvd
established
ACK_a, ack_a, win_a
established
Estab
Estab

Figure 7. Message sequence chart for the TCP connection establishment.

Figure 7 shows the general procedure for TCP connection establishment in which each segment has several factors. We need not assign an initial value of acknowledgement number because TCP acknowledges the other TCP's SYN by ACKing the other's ISN plus one. Since the acknowledgement number and the next sequence number are fixed by the initial sequence number and the next window size maintains the default, each tester sends initial values of three factors to TCP.

(2) As shown in Figure 7, Tester A (Tester B) assigns initial values of factors to TCP A (TCP B) respectively. When 'active_open', 'passive_open', or 'send_data' message is given by the application program running on TCP, values of the three factors in the TCP packets are initialized. When the packets between TCPs are exchanged, these values are transmitted to the other TCP.

(3) All factors have two levels and levels of the factors are shown in Table 3.

Because the level space is usually very large, it is impossible to sample all level values. So in order to provide better sampling coverage, levels need to be chosen properly. The method to choose levels follows the three principles [9]: (i) for the level space, it is better to choose the minimum and maximum values and to partition it into homogeneous spaces for middle values, (ii) levels are better to include values used in current systems and expected to be the optimal solution, and (iii) the suitable size of level space is from 2 to 5 because over 5 level space makes the domain of factors be hard to be managed. In this work, for the sequence number, we partition the level space into homogeneous spaces and then choose values including the minimum and maximum. For the window size and the maximum segment size, we choose values used in current systems. These values of each factor can be assigned to the tester by the implementer through the Protocol Implementation Conformance Statement / Protocol Implementation eXtra Information for Testing (PICS/PIXIT).

Table 3. Levels for factors.

Factor Level	seq of TCP A	win of TCP A	mss of TCP A	seq of TCP B	win of TCP A	mss of TCP A
1	2823083521	2048	1460	1415531521	4096	1024
2	0	8192	256	4294967295	16384	256

3.2 Test Suite Generation

To generate the test suite for the data part, we consider the assumption (1) and (2) (Condition 1). Then the derived test suite has two types. For example, the item <1> of Appendix 1 which is an example of test case for the message interaction in Figure 7 is the first type and as follows:

(Closed,Listen) -- (active_open_a,seq,win,mss)/[<,(i_SYN,seq,win,mss),, (i_SYN_ACK,seq,ack,win,mss)>, <established,(i_ACK,ack,win), established,>] → (Estab,Estab)

Judging from the fact that (Closed,Listen) is the starting stable state, TCP B has already received 'passive_open' message and initial values of its factors from Tester B. TCP A receives 'active_open' message and initial values of it's factors from Tester A. Thus this example of test case is a test case with six factors because both TCPs use three factors. Like this, in generating the test suite for the data part, eight test cases with six factors are represented by the items <1> to <8> of Appendix 1.

The item <9> of Appendix 1, for example, is the second type. It is an example of test case for the message interaction in Figure 8 and as follows:

(Closed,Closed) -- (active_open_a,seq,win,mss)/[<,(i_SYN,seq,win, mss),,>] → (SYN_Sent,Closed)

When TCP B is not ready to receive packets from TCP A because of not receiving 'passive_open' message from Tester B, TCP B is in state Closed. Thus this test case have only three factors. In generating the test suite for the data part, four test cases with three factors like this is represented by the items <9> to <12> in Appendix 1.

Let us calculate the size of test suite after choosing all possible values of the factors. The 32-bit sequence number field has 2^{32} levels and the 16-bit window size and the 16-bit maximum segment size fields have 2^{16} levels. The eight test cases with six factors have 2^{128} (= $2^{32}\text{x}2^{32}\text{x}2^{16}\text{x}2^{16}\text{x}2^{16}\text{x}2^{16}$) test cases for the data part respectively and the four test cases with three factors have 2^{64} (= $2^{32}\text{x}2^{16}\text{x}2^{16}$) test cases respectively. Thus the total number of test cases for the data part is $2^{131}+2^{66}$ (= $8\text{x}2^{32}\text{x}2^{32}\text{x}2^{16}\text{x}2^{16}\text{x}2^{16}\text{x}2^{16}+4\text{x}2^{32}\text{x}2^{16}\text{x}2^{16}$).

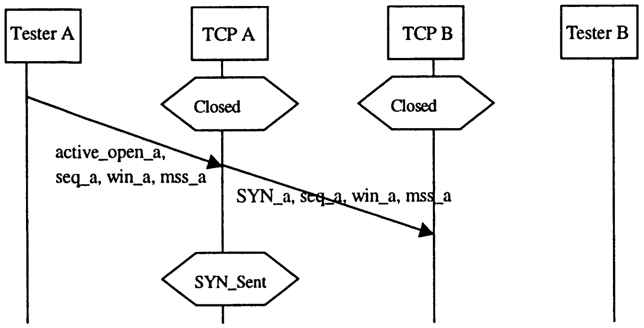
Tester A
TCP A
TCP B
Tester B
Closed
Closed
active_open_a, seq_a, win_a, mss_a
SYN_a, seq_a, win_a, mss_a
SYN_Sent

Figure 8. Message sequence chart in case of using only factors of one TCP.

Let us calculate the size of test suite after considering the assumption (3) (Condition 2) as well as the assumptions (1) and (2) (Condition 1). The eight test cases with six factors have 64 (= 2^6) test cases for the data part respectively and the four test cases with three factors have 8 (= 2^3) test cases respectively. Thus the total number of test cases is 544 (= $8\times2^6+4\times2^3$).

3.3 Test Suite Generation using Experimental Design

The size of test suite derived in Section 3.2 can be reduced by using orthogonal arrays (Condition 3). Table 4 represents 2-level orthogonal array substituting 2 for m and we randomly arrange six factors to the columns. We chose the column numbers 1 to 6 for the arrangement of the factors. The arrangement of levels of each factor, 1 and 2, is randomly decided and follows Table 3 in this paper. Table 4 means that it is possible to test interoperability for the TCP data part with only 8 test cases without 64 (= 2^6) with all combinations of test factors.

Table 4. Orthogonal array for 6 factors, 2 levels.

Experimental number	Column number (factor)						
	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2
Arrangement	seq of TCP A	win of TCP A	mss of TCP A	seq of TCP B	win of TCP B	mss of TCP B	

For the eight test cases with six factors, we arrange their factor levels by using the orthogonal array in Table 4 and for the four test cases with three factors by using the orthogonal array in Table 1. In case of using Table 1, the sequence number, the window size, and the maximum segment size are arranged in the column numbers 1, 2, and 3 respectively. Therefore the total number of test cases is 80 (= 8x8+4x4). Appendix 2 shows eight and four test cases respectively from the items <1> and <9> of Appendix 1. This test case for the item <1> of Appendix 2 has levels assigned by the experimental number 1 of Table 4.

(Closed,Listen) -- (active_open_a seq=2823083521,win=2048, mss=1460)/[<,(i_SYN,seq=2823083521,win=2048, mss=1460),, (i_SYN_ACK,seq=1415531521,ack=2823083522,win=4096,mss=1024)

>, <established,(i_ACK_a,ack=1415531522,win=2048),established,>]
$\rightarrow$ (Estab,Estab)

TCP A sends SYN segment with its sequence number 2823083521, window size 2048, and maximum segment size 1460 to TCP B. TCP B responds with its own SYN segment containing TCP B's sequence number 1415531521, window size 4096, and maximum segment size 1024. TCP B also acknowledges TCP A's SYN by ACKing TCP A's ISN plus one. TCP A acknowledges this SYN from TCP B by TCP B's ISN plus one. So this is the test case to establish connection by the three-way handshake.

3.4 Assessment

Table 5 shows size of test suite for the data part when we add each of three conditions in order. By Condition 1, the size of test suite becomes $2^{131}+2^{66}$ and by the addition of Condition 2, it becomes 544. Also by Condition 3 using orthogonal arrays, the size of it is reduced to 80 while a well-defined level of test coverage is maintained in terms of the observation illustrated in Section 2.2. Thus compared to (B), there is an 85% reduction in the number of test cases and compared to (A), there is a reduction of more than 99.999%. Because most field faults are caused by the interactions of two factors, 80 test cases covering all pairwise combinations have nearly the same test coverage, compared to the 544 test cases.

Table 5. Comparison of test suite size at each stage.

Indication of Figure 6	Conditions for generating the test suite for data part	Size of test suite	Note
(A)	Condition 1. Considering the assumptions (1) and (2)	$2^{131}+2^{66}$	(C) : 85% reduction compared to (B) and reduction of more than 99.999% compared to (A)
(B)	Condition 2. Considering the assumption (3)	544	
(C)	Condition 3. Using orthogonal arrays of Experimental Designs	80	

4. CONCLUSION AND FUTURE WORK

In this paper, we presented the test generation method suitable for testing interoperability of the data part for TCP connection establishment. For this work, we laid down three assumptions for the data part and considered experimental designs in order to reduce the size of the test suite while maintaining a well-defined level of test coverage. The 80 test cases were finally generated for testing the data part. Thus compared to the case of choosing all possible values as the level space, there is a reduction of more than 99.999% in the number of test cases and compared to the case of reducing the level space, there is an 85% reduction. This method leads to a faster detection of non-interoperation, which would help to get a higher quality of products in a shorter development interval.

As further work, this method will be applied to the TCP connection release and data transmission phases. We need to develop an algorithm for generating the test suite for the data part and to implement the algorithm. Also we will demonstrate the feasibility of the algorithm by comparing its application result with the test suite derived manually in this paper and the generality of the algorithm by applying it to the other protocols.

REFERENCES

[1] K. Burroughs, A. Jain, and R. L. Erichson, "Improved Quality of Protocol Testing Through Techniques of Experimental Design," Supercomm/ICC '94, 1994.

[2] D. M. Cohen, S. R Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design," IEEE Transactions on Software Engineering, Volume 23, Number 7, pp.437-444, July 1997.

[3] D. M. Cohen, S. R. Dalal, A. Kajla, and G. C. Patton, "The Automatic Efficient Test Generator (AETG) System," Proc. 5th Int'l Symp. Software Reliability Eng., IEEE CS Press, pp. 303-309, 1994.

[4] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton, "The combinatorial design approach to automatic test generation," IEEE Software Volume: 13, Issue: 5, pp. 83-88, September 1996.

[5] ISO/IEC/9646, OSI Conformance Testing Methodology and Framework Parts 1-7, 1994.

[6] ITU-T X.290 Series, Conformance Testing Methodology and Framework, 1994.

[7] V. Jacobson, C. Leres, and S. McCanne, The Tcpdump Manual Page, Lawrence Berkeley National Laboratory, Berkeley, CA., June 1997.

[8] S. Kang and M. Kim, "Interoperability Test Suite Derivation for Symmetric Communication Protocols," IFIP Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE X) and Protocol Specification Testing and Verification (PSTV XVII), pp. 57-72, November 1997.

[9] D. C. Montgomery, Design and Analysis of Experiments, 4th Ed., John Wiley & Sons, Inc., 1997.

[10] G. S. Peace, Taguchi Methods: A Hands-On Approach, Addison-Wesley, 1993.

[11] J. B. Postel, "Transmission Control Protocol," RFC 793, September 1981.

[12] W. Richard Stevens, TCP/IP Illustrated, Volumel: The Protocols, Addison-Wesley, 1995.

[13] S. Seol, M. Kim, S. Kang, and Y. Park, "Interoperability Test Suite Derivation for the TCP," IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX), October 5-8, Beijing, China, 1999.

[14] J. Shin and S. Kang, "Interoperability Test Suite Derivation for the ATM/B-ISDN Signaling Protocol," Testing of Communicating Systems, Vol. 11, Kluwer Academic Publishers, pp. 313-330, 1998.

[15] A. W. Williams and R. L. Probert, "A Practical Strategy for Testing pair-wise Coverage of Network Interfaces," Proceedings of the 7th International Conference on Software Reliability Engineering (ISSRE '96), White Plains NY USA, pp. 246-254, October 1996.

Appendix 1. Test suite related to the control part of TCP connection establishment.

<1> (Closed,Listen) -- active_open_a/[<,i_SYN,,i_SYN_ACK>, <established,i_ACK,established,>] → (Estab,Estab)
<2> (Listen,Closed) -- active_open_b/[<,i_SYN_ACK,,i_SYN>, <established,,established,i_ACK >] → (Estab,Estab)
<3> (Closed,SYN_Sent) -- active_open_a/[<,i_SYN,,i_SYN_ACK>, <established,i_ACK,established,>] → (Estab,Estab)
<4> (SYN_Sent,Closed) -- active_open_b/[<,i_SYN_ACK,,i_SYN>, <established,,established,i_ACK>] → (Estab,Estab)
<5> (Listen,Listen) -- send_data_a/[<,i_SYN,,i_SYN_ACK>, <established,i_ACK,established,>] → (Estab,Estab)
<6> (Listen,Listen) -- send_data_b/[<,i_SYN_ACK,,i_SYN>, <established,,established,i_ACK >] → (Estab,Estab)
<7> (Listen,SYN_Sent) -- send_data_a/[<,i_SYN,,i_SYN_ACK>, <established,i_ACK,established,>] → (Estab,Estab)
<8> (SYN_Sent,Listen) -- send_data_b/[<,i_SYN_ACK,,i_SYN>, <established,,established,i_ACK >] → (Estab,Estab)
<9> (Closed,Closed) -- active_open_a/[<,i_SYN,,>] → (SYN_Sent,Closed)
<10> (Closed,Closed) -- active_open_b/[<,,,i_SYN>] → (Closed,SYN_Sent)
<11> (Listen,Closed) -- send_data_a/[<,i_SYN,,>] → (SYN_Sent,Closed)
<12> (Closed,Listen) -- send_data_b/[<,,,i_SYN>] → (Closed,SYN_Sent)

Appendix 2. Test suite for the TCP data part from the items <1> and <9> in Appendix 1.

<1> (Closed,Listen) -- (active_open_a,seq=2823083521, win=2048,mss=1460)/ [<,(i_SYN,seq=2823083521,win=2048,mss=1460),,(i_SYN_ACK, seq=1415531521,ack=2823083522,win=4096,mss=1024)>, <established,

(i_ACK,ack=1415531522,win=2048),established,>] → (Estab,Estab)

<2> (Closed,Listen) -- (active_open_a,seq=2823083521, win=2048,mss=1460)/ [<,(i_SYN,seq=2823083521,win=2048,mss=1460),,(i_SYN_ACK, seq=4294967295,ack=2823083522,win=16384,mss=256)>, <established, (i_ACK,ack=0,win=2048),established,>] → (Estab,Estab)

<3> (Closed,Listen) -- (active_open_a,seq=2823083521, win=8192,mss=256)/ [<,(i_SYN,seq=2823083521,win=8192,mss=256),,(i_SYN_ACK, seq=1415531521,ack=2823083522,win=4096,mss=256)>,<established, (i_ACK,ack=1415531522,win=8192),established,>] → (Estab,Estab)

<4> (Closed,Listen) -- (active_open_a,seq=2823083521, win=8192,mss=256)/ [<,(i_SYN,seq=2823083521,win=8192,mss=256),,(i_SYN_ACK, seq=4294967295,ack=2823083522,win=16384,mss=1024)>, <established, (i_ACK,ack=0,win=8192),established,>] → (Estab,Estab)

<5> (Closed,Listen) -- (active_open_a,seq=0,win=2048,mss=256)/[<,(i_SYN, seq=0,win=2048,mss=256),,(i_SYN_ACK,seq=1415531521,ack=1, win=16384,mss=1024)>, <established,(i_ACK,ack=1415531522,win=2048), established,>] → (Estab,Estab)

<6> (Closed,Listen) -- (active_open_a,seq=0,win=2048,mss=256)/[<,(i_SYN, seq=0,win=2048,mss=256),,(i_SYN_ACK,seq=4294967295,ack=1,win=4096, mss=256)>, established,(i_ACK,ack=0,win=2048),established,>] → (Estab,Estab)

<7> (Closed,Listen) -- (active_open_a,seq=0,win=8192,mss=1460)/[<,(i_SYN, seq=0,win=8192,mss=1460),,(i_SYN_ACK,seq=1415531521,ack=1, win=16384,mss=256)>, <established,(i_ACK,ack=1415531522,win=8192), established,>] → (Estab,Estab)

<8> (Closed,Listen) -- (active_open_a,seq=0,win=8192,mss=1460)/[<,(i_SYN, seq=0,win=8192,mss=1460),,(i_SYN_ACK,seq=4294967295,ack=1, win=4096,mss=1024)>, <established,(i_ACK,ack=0,win=8192), established,>] → (Estab,Estab)

<9> (Closed,Closed) -- (active_open_a,seq=2823083521,win=2048,mss=1460)/ [<,(i_SYN,seq=2823083521,win=2048,mss=1460),,>] → (SYN_Sent,Closed)

<10> (Closed,Closed) -- (active_open_aseq=2823083521,win=8192,mss=256)/ [<,(i_SYN,seq=2823083521,win=8192,mss=256),,>] → (SYN_Sent,Closed)

<11> (Closed,Closed) -- (active_open_a,seq=0,win=2048,mss=256)/ [<,(i_SYN,seq=0,win=2048,mss=256),,>] → (SYN_Sent,Closed)

<12> (Closed,Closed) -- (active_open_a,seq=0,win=8192,mss=1460)/ [<,(i_SYN,seq=0,win=8192,mss=1460),,>] → (SYN_Sent,Closed)