Mutation Testing of Protocol Messages Based on Extended TTCN-3*

Chuanming Jing\textsuperscript{1,2}, Zhiliang Wang\textsuperscript{1,3}, Xingang Shi\textsuperscript{1,3}, Xia Yin\textsuperscript{1,2}, Jianping Wu\textsuperscript{1,2,3}

\textsuperscript{1} Tsinghua National Laboratory for Information Science and Technology
Beijing, P. R. China, 100084

\textsuperscript{2} Department of Computer Science & Technology, Tsinghua University
Beijing, P. R. China, 100084

\textsuperscript{3} Network Research Center of Tsinghua University
Beijing, P. R. China, 100084

\{jcm, wzl, shixg, yxia\}@csnet1.cs.tsinghua.edu.cn

Abstract

The critical requirement on reliability, fault-tolerance and security of network devices highlights the necessity of protocol robustness testing. Mutation testing of protocol messages is an important part of robustness testing, but related theory and practices are not well developed. This paper builds a NFSM model for mutation testing of protocol messages and proposes two types of Normal-Verification Sequence to enhance verdict mechanism. For single-field mutation testing of protocol messages, we propose the concept of compound anomalous test case to further simplify test sequences. As a standard test specification language, TTCN-3 reveals strong excellence in conformance testing, so we apply TTCN-3 to mutation testing and extend it according to test requirements. Using our method we test OSPFv2 sufficiently with a test system based on extended TTCN-3. The results indicate that our method has good capability of error-finding.

1. Introduction

Protocol specification commonly prescribes how to process valid inputs as well as a few invalid inputs. As conformance testing only verifies whether an implementation conforms to its protocol specification or not, the capability of error-detection is limited. Additionally, the scale and complexity of internet increase greatly and there exists noise, disturbance, misconfiguration and vicious man-made attacks in the internet. The critical requirement on reliability, fault-tolerance and security of network devices highlights the necessity of protocol robustness testing.

Protocol robustness testing is the test to verify whether IUT (Implementation under Test) can function correctly in the presence of invalid inputs or stressful environmental conditions [1].

Invalid inputs include messages with invalid syntax (i.e. messages which disobey protocol specification data formats) and messages with anomalous semantics (i.e. messages which have valid syntax but conflict with protocol state, configuration, parameters and policies…). Robustness testing by injecting messages with invalid syntax is called mutation testing of protocol messages. It aims to detect vulnerabilities of protocol specification and protocol implementations, including [2]:

- The vulnerability of malformed message parsing;
- The vulnerability of state transition;
- The vulnerability of huge resource consumption when lots of anomalous messages are injected;
- The hole of buffer overflow.

Related works about mutation testing of protocol messages are as follows: PROTOS [3,4] is a pioneering project. The method is that tester describes the structure of PDU with BNF (Backus-Naur Form), writes the mutation strategy in configuration files and then uses a tool to read the configuration files and further produce anomalous messages.

[5-9] all follow the method of PROTOS, but the descriptions of PDU (Protocol Data Unit) are different. [5,6] make use of SBNF (Strengthened BNF); [7,8] adopt SCL (Semantic Constraint Language) and [9] uses XML. The production and injection of anomalous PDUs are all done by tools implemented by C or Java.

Many network engineers would like to test protocol implementations using Fuzzing Tools [10] which can produce many random protocol messages. But these randomly produced messages are blind to testing and each tool can only test a certain protocol. The test verdict is also simple and not so reasonable because they only observe whether IUT is down under anomalous injection.

The study of [11] belongs to robustness testing, but they do not refer to mutation testing of protocol messages. [12] studies the robustness testing of BGP protocol, but most of the test cases are used to test semantics of protocol specification. The types of mutation operations of protocol messages are very limited.
All these test practices have found out vulnerabilities of protocol implementations. But their shortages are distinct:

- The design of anomalous test cases lacks guidance of theory;
- The verdict mechanism needs improvement;
- The structure of test case is not optimal, resulting in great workload;
- All focus on a certain protocol, and the test system is not general for other protocols.
- All use programming languages such as C and Java to build test suite, so the readability, scalability and maintainability are not good.

To cope with these deficiencies, we propose a formal model for mutation testing of protocol messages and two types of Normal-Verification Sequence to improve verdict mechanism. For single-field mutation testing of protocol messages, the concept of compound anomalous test case is proposed to simplify test sequences.

As a standard test specification language, TTCN-3 [15] can be applied in almost all kinds of black-box testing of reactive and distributed systems, so we apply TTCN-3 to mutation testing and extend it according to test requirements.

The rest of this paper is organized as follows. Section 2 proposes a formal model for mutation testing of protocol messages and two types of Normal-Verification Sequence based on this model. In section 3, we propose the concept of compound anomalous test case for single-field mutation testing of protocol messages. We apply TTCN-3 to mutation testing and extend it according to test requirements in section 4. In section 5, we test OSPFv2 [17] using our method and results are given. Conclusions and future work are given in Section 6.

2. Formal Model

2.1. Model definitions

A PDU is composed of several fields denoted as \( f_1, f_2, \ldots \). We define anomalous PDU as follows.

**Definition 1: Anomalous PDU**

A PDU \( P = < f_1, f_2, \ldots, f_n > \) is anomalous, iff, \( \exists i \in [1,n] \), such that \( f_i \) disobeys Protocol Specification Data Formats.

Conformance testing often builds formal model using FSM (Finite State Machine). As most anomalous PDUs and their processing rules are not prescribed in protocol specification, the state transitions after these anomalous injections are nondeterministic. So we build a formal model for mutation testing of protocol messages using NFSM (Nondeterministic Finite State Machine) [13].

**Definition 2: NFSM Model for Mutation Testing of Protocol Messages**

Protocol \( P \) can be specified as NFSM, \( P = < I, O, S, T, s_0 > \), where

\( I \) : a set of inputs and \( I = I_{spec} \cup I_{unspec} \), \( I_{spec} \) includes inputs that are prescribed in protocol specification and composed of valid PDUs and a few anomalous PDUs. \( I_{unspec} \) includes inputs that are not prescribed in protocol specification and composed of lots of anomalous PDUs.

\( S \) : a set of states, \( S = S_{spec} \cup S_{unspec} \), \( S_{spec} \) includes states prescribed in protocol specification. We define \( S = \{ s_1, s_2, \ldots \} \), \( s_1, s_2, \ldots \) are all nondeterministic states but within a range of states according to the ambiguous protocol specification, i.e. \( s_j \in S \subseteq S_{spec} \), \( s_{j+1} \in S \subseteq S_{spec} \), So, \( S = S_{spec} \cup S_{unspec} \). \( s_0 \) : initial state.

\( O \) : a set of outputs.

\( T \) : a set of state transitions, \( T = T_{spec} \cup T_{nondet} \),

\[ T_{spec} = \bigcup t_{l_j} : s_j \xrightarrow{i_j,s_j \in S_{spec}} s_{j+1} \], \( i_j \in I_{spec} \);

\[ o_j \in O \ or \ o_j = \text{Null} \ . \ T_{nondet} \ = \bigcup t_{l_j} : s_j \xrightarrow{i_j,s_j \in S_{unspec}} s_{j+1} \],

where \( s_j \in S_{spec}, s_{j+1} \in S_{spec} \); \( i_j \in I_{unspec} \); output is unknown or nondeterministic.

We emphasize that, if \( i_j \in I_{spec} \), and the state transition of \( s_j \) after receiving \( i_j \) is not defined by protocol specification, we do not think this kind of transition belongs to \( T_{nondet} \) of our NFSM model which is defined for mutation testing of protocol messages.

Figure 1 shows two kinds of state transitions under anomalous injections. Figure 1(a) shows after receiving \( i_j \ (i_j \in I_{spec}) \), \( s_j \) transits to \( s_{j+1} \) according to related description in protocol specification. Figure 1(b) shows that \( s_j \) transits to unknown state \( s_{j+1} \) because the state transition after receiving \( i_j \ (i_j \in I_{unspec}) \) is not prescribed or prescribed nondeterministicly in protocol specification.

Figure 1. (a) Figure1. (b) Figure 2.

Figure 1. State transitions prescribed determinately and nondeterministically in protocol specification

Figure 2. Forced State Transition of \( s_{j+1} \)

According to protocol specification, there exits some special inputs which can transit each of several states.
(denoted as $S'$) to the same state. So we define a typical type of state transition as follows:

**Definition 3: Forced State Transition**

Let $S' \subseteq S_{\text{spec}}$ and $s_j \in S_{\text{spec}}$.

\[ \forall s \in S', \ s \rightarrow s_j \text{ is a Forced State Transition, iff, } \exists i, \text{ such that } \forall s \in S', \ s \xrightarrow{i/o_j} s_j. \]

Especially, if $s_j = s_0$, this state transition is also called State Transition Reset.

In our NFSM model, unknown state $s_{rk} \in S$, and $s_{rk} \in S' \subseteq S_{\text{spec}}$. According to definition 3, $s_{rk}$ can receive some inputs and transit to a deterministic state. It is shown in Figure 2.

For OSPF protocol, event “1-Way” forces the NFSM under state of “2-Way” or higher to transit to “Init” state and event “SeqNumberMismatch” forces the NFSM under state of “Exchange” or higher to transit to “ExStart” state.

For BGP protocol, receiving “Notification” message can cause State Transition Reset. For SIP protocol, receiving “Cancel” can cause State Transition Reset.

[13] gives a survey about conformance testing based on FSM. The structure of a test case in conformance testing can be described as follows:

**Test Case:** A Test Case=<State Leading Sequence, Executing Sequence, State Verification Sequence>

State Leading Sequence brings FSM into state (denoted as $s_j$). Executing Sequence can apply input (denoted as $i_j$) and observe the output. State Verification sequence aims to verify the state (denoted as $s_j$).

In mutation testing, we introduce a term called “anomalous test case” which can inject anomalies and make verdict. Instead of State Verification Sequence of conformance testing, Normal-Verification Sequence of mutation testing is executed to verify whether the state machine works properly. If it returns “Fail”, we conclude that IUT behaves abnormally and has poor robustness. The structure of an anomalous test case can be described as follows:

**Anomalous Test Case:** An Anomalous Test Case=<State Leading Sequence, Anomalous PDU Inputting, Normal-Verification Sequence>

We discuss the construction of Normal-Verification Sequence in Section 2.2.

### 2.2. Two types of Normal-Verification Sequence

No existing mutation testing methods considers how to construct valid and reasonable Normal-Verification Sequence. They only observe whether IUT is down under anomalous injection. The construction of Normal-Verification Sequence is depended on robustness requirement. In this paper, we use an intuitive and simple robustness requirement that IUT must keep normal under anomalous injection and must conform to protocol specification.

Considering the state transitions shown in Figure 1 and Figure 2, we propose two types of Normal-Verification Sequence as follows:

**Normal-Verification Sequence 1:**

If an anomalous PDU $i \in I_{\text{spec}}$ and $s_j \xrightarrow{i/o} s_j$, $o \in O$ or $o$ = Null, $s_j \in S_{\text{spec}}$. Normal-Verification Sequence can be constructed by applying state verification sequence of $s_j$.

**Normal-Verification Sequence 2:**

If an anomalous PDU $i \in I_{\text{nosp}}$ and $s_j \xrightarrow{i/o} s_{rk}$, $o \in O$ or $o$ = Null, $s_j \in S_{\text{spec}}, s_{rk} \in S$. Normal-Verification Sequence can be constructed with state identification sequence of $s_{rk}$.

However, Normal-Verification Sequence 2 is hard to be used in test practice because identification of nondeterministic state in NFSM model is a very difficult problem to resolve [13]. Moreover, thousands of anomalous messages will be injected in mutation testing so state identification will make the test sequences more complex and eventually consume huge workload. In the end, mutation testing of protocol messages cares more about whether IUT behaves normally than state transition after anomalous injection, which is different from conformance testing of NFSM [13,14].

According to the transition of unknown state $s_{rk}$, we propose other Normal-Verification Sequence to approximately replace Normal-Verification Sequence 2. We make use of Forced State Transition and define:

**Normal-Verification Sequence 2-1:**

After anomalous PDU injection, the state is $s_{rk} \in S$, and $s_{rk} \in S' \subseteq S_{\text{spec}}$, we can make use of Forced State Transition: $s_{rk} \rightarrow s_j$, then apply state verification sequence of $s_j$.

Now we take OSPF as an example to construct the Normal-Verification Sequence. According to OSPF specification, after receiving most (not all) anomalous Link State Update messages under state “Full”, the state remains “Full”. Normal-Verification Sequence 1 can be used to construct anomalous test case in this situation.

During database synchronization we suppose state transits to $s_{rk}$ after anomalous injection. We can use event “SeqNumberMismatch” to force state to $s_j$ (ExStart state) and apply state verification sequence of $s_j$. This is an example of Normal-Verification Sequence 2-1.

Other than Normal-Verification Sequence 2-1, we can also build other types of Normal-Verification Sequence.
such as just an input and the received reply to make weaker verdict. We take OSPF as an example: at any state, state machine must process Hello message and reply with a Hello message which can be used to make verdict. It should be noticed this verdict is not strict although it is convenient to use.


In mutation testing of protocol messages, if only one field of anomalous PDU is mutated, it is also called single-field mutation testing of protocol messages. In this section, we study single-field mutation testing of protocol messages and propose the concept and structure of compound anomalous test case.

3.1. Algorithm of anomalous test case generation

In test practice, it is impossible to inject all anomalous messages. So we define some typical anomalous field values. We sum up some typical anomalous messages attackers tend to use as follows:

- Anomalous field value
  - Boundary value: maximum value; maximum value+1; minimum value; minimum value-1; invalid value…
  - Field values mismatch: mutation of field such as “Type” and “Version”.
  - Format error: fields such as “Network”, “Network mask” and so on have special format.
- Field Mutation
  A filed of PDU is added or removed; one field is replaced with another field with bigger bytes and causes overflow; location of one field changes.
- Length, Checksum and Encapsulation error
  The values of fields such as “length”, “checksum” have mistakes. For protocols such as SIP and SNMP, the Separate Symbol and BER (Basic Encoding Rules) encapsulation may also bring errors.

Then we give the algorithm of anomalous test case generation for single-field mutation testing of protocol messages called Loop Replace Method() in Figure 3. Figure 3 also give the algorithm of test group generation.

3.2. Compound Anomalous Test Case

According to test practice of PROTOS [3,4], when several test cases are executed in sequence continuously the pass rate is lower than that when they are executed independently. This indicates that IUT is easier to be abnormal under injections of a number of anomalous messages. Robustness testing needs injecting lots of anomalous messages. Also, if several test cases are executed continuously, subsequent test case need not lead the state from initial state. Otherwise, there will be redundancy.

Considering the requirement of robustness testing and the method of simplifying test sequences, we propose the concept of compound anomalous test case. In this paper, we only apply compound anomalous test case to single-field mutation testing of protocol messages.

![Figure 3. Algorithm of anomalous test case generation: Loop Replace Method()](image)

Compound anomalous test cases have features such as:

- One compound anomalous test case focuses on only one field (denoted as $f_i$) of PDU. If the verdict is “Fail”, it means IUT cannot parse $f_i$ with robustness.
- Two or more anomalous messages with different anomalous values of $f_i$ should be injected in one compound anomalous test case. Values of other fields of these messages cannot be mutated and keep valid.
- Test sequences can be simplified in compound
anomalous test case.

According to the types of Normal-Verification Sequence, we propose two kinds of compound anomalous test cases:

1) Compound Anomalous Test Case-1

For anomalous test case using Normal-Verification Sequence_1, after receiving an anomalous PDU under state \( s_0 \), the state will transit to \( s_1 \). Making use of this feature of state transition we propose the structure of Compound Anomalous Test Case-1 as follows:

**Compound Anomalous Test Case-1**

\[ \langle \text{State } (s_0 \text{ to } s_1) \text{ Leading Sequence, } \{ \text{Anomalous PDU Inputting, State}(s_0 \text{ to } s_1) \text{ Leading Sequence } \}^+, \text{ State Verification Sequence of } s_1 \rangle. \]

The structure of Compound Anomalous Test Case-1 is shown in Figure 4. From this structure, we can get its generation algorithm by modifying Loop_Replace_Method. The algorithm is shown in Figure 5.

![Figure 4. Compound Anomalous Test Case-1 (PDU.field is under test)](image)

We give an example using OSPF, after receiving most (not all) anomalous Link State Update messages under state “Full”, the state remains “Full” and then next anomalous Link State Update messages can be injected...So we can construct Compound Anomalous Test Case-1 and \( s_i = s_j = \text{\textit{Full}} \), State \( (s_j \text{ to } s_i) \text{ Leading Sequence}=\text{Null}. \)

2) Compound Anomalous Test Case-2

For anomalous test case using Normal-Verification Sequence_2-1, we propose:

**Compound Anomalous Test Case-2**

\[ \langle \text{State } (s_0 \text{ to } s_1) \text{ Leading Sequence, } \{ \text{Anomalous PDU Inputting, State}(s_0 \text{ to } s_1) \text{ Leading Sequence (using Forced State Transition), State}(s_0 \text{ to } s_1) \text{ Leading Sequence } \}^+, \text{ State Verification Sequence of } s_1 \rangle. \]

Input:

\[
PDU \ pdu = \langle f_1, f_2, \ldots, f_n \rangle; \quad E_{\ pdu} = \{ e_1, e_2, \ldots, e_m \};
\]

Output:

\[
TestGroup_{pdu} ; \quad // \text{ Test Group for } pdu
\]

\[
TestCase_{f_i} ; \quad // \text{ Compound anomalous test case-1}
\]

Initial Value:

\[
TestGroup_{pdu} = \text{TestCase}_{f_i} = \text{Null};
\]

// Please note that \( i \) and \( j \) are different, also \( j \) and \( j \).

1. **For each** \( f_i \in pdu \)
2. \[
\text{Valid } _\text{FieldValue} = \ pdu \ p_i ;
\]
3. **TestCase}_{f_i} .add(\text{State } (s_0 \text{ to } s_i) \text{ Leading Sequence});
4. **For each** \( e_j \in E_{f_i} \)
5. replace \( pdu \ f_i \text{ with } e_j ;
6. **TestCase}_{f_i} .add (\text{Anomalous pdu Inputting, State } (s_j \text{ to } s_i) \text{ Leading Sequence});
7. **End For**
8. **TestCase}_{f_i} .add(\text{State Verification Sequence of } s_j);
9. \[
\text{TestGroup}_{pdu} \leftarrow (\text{TestGroup}_{pdu} \cup \text{TestCase}_{f_i});
\]
10. \[
pdu = \text{Valid } _\text{FieldValue};
\]
11. **End For**

Figure 5. Algorithm of Compound Anomalous Test Case-1 generation

Figure 6 shows the structure of Compound Anomalous Test Case-2. We can get its generation algorithm by modifying Loop_Replace_Method easily. Due to limited space we omit it.

![Figure 6. Compound Anomalous Test Case-2 (PDU.field is under test)](image)

For OSPF, during database synchronization, the state transits to unknown state after anomalous message injection. We can use event “SeqNumberMismatch” to force state to “Exstart” and then lead state to “Exchange” to inject next anomalous PDU...
Although compound anomalous test case simplifies test sequences by combining several separated test cases, the capability of error-detection is not weaken. Our reason is that if verdict of one separated test case is “Fail”, the compound anomalous test case will also make verdict of “Fail”. In fact, it is more effective to test the robustness of IUT by injecting several anomalous messages in one compound anomalous test case. Also, for the compound test case which returns “Fail”, we should decompose it into several test cases executed further to analyze why it fail and to discover which anomalous message causes it. As the pass rate is often very high in robustness testing, the decomposition and test execution of separated test cases will not consume much work.

4. Extension of TTCN-3

TTCN-3 [15] is developed by ETSI (European Telecommunications Standards Institute). We have developed a test system called PITSv3 [16] based on TTCN-3 for conformance testing. In this section, we will extend TTCN-3 based on the requirements of mutation testing. Due to limited space, we do not discuss the extension of PITSv3 in detail in this paper.

4.1. Using TTCN-3 in mutation testing

TTCN-3 has many advantages including: completely dynamic test configurations; support synchronous communication; support testing of distributed systems; strong message matching mechanism and so on. The test suite described by TTCN-3 has good readability, scalability and maintainability. TTCN-3 has been used in many types of testing.

The first step of constructing a test suite composed of compound anomalous test cases is to define some anomalous field sets such as \( E_i = \{ e_1, e_2, \ldots, e_m \} \). As an example, we give the boundary value set of 8Bit-field described in format of “const record” as follows:

```plaintext
type record Boundary8bit {
  Oct1 Boundary8bit1,
  Oct1 Boundary8bit2,
  Oct1 Boundary8bit3,
  Oct1 Boundary8bit4,
  Oct1 Boundary8bit5
}
const Boundary8bit Boundary8bit_value := {
  Boundary8bit1 := 'ff'O,
  Boundary8bit2 := '00'O,
  Boundary8bit3 := 'fe'O,
  Boundary8bit4 := '01'O,
  Boundary8bit5 := '7f'O
}
```

Because these data are typical and representative, the test effect is good. Also, every anomalous field value set can be used to replace many fields of various messages. For example, every field with 8 bits of all messages can be replaced with “Boundary8bit_value” in boundary value testing. The field with 8 bits can also be replaced with “Boundary64bit_value” in overflow testing.

Then we can use TTCN-3 to describe the algorithm proposed in section 3. But comparing to specification language, TTCN-3 is not flexible enough to achieve this function:

- TTCN-3 cannot support making mutation operation well.
- Anomalous messages injected are commonly described by “global template” in TTCN-3 and need modifying frequently. But TTCN-3 cannot support these well. First, keyword “Modifies” can not modify one template directly but generate another, so the number of data will increase largely. Second, “dot notation” shall not be used to set values in the format of “global template”. Third, although we can achieve this purpose using template parameter, it is not convenient because almost every field will be mutated and lots of template parameters should be defined.
- It is complex and difficult to describe the algorithm of compound anomalous test case in TTCN-3.

TTCN-3 is not good at describing compound anomalous test cases. For example, anomalous field values can be defined using “array”, “record of” or “set of”, then the number of elements in the record can be gained using TTCN-3 predefined function “sizeof”; using “for” loop statement each anomalous field value can be retrieved from anomalous values set in format of “array” or other and can be used to mutate a valid message in format of “global template”. Additionally, valid field value must be saved before mutation and be assigned to the field under test after mutation and injection in order to revert to valid value and insure single-field mutation.

If we do not extend TTCN-3, the description statements will be complex.

4.2. Extension of TTCN-3 and its test system

Considering above reasons, we extend TTCN-3 in syntax as follows:

- Add new keyword “loopreplace” which represents the complex algorithm of compound anomalous test case generation, the corresponding description statement is quite simple.
- In the description statement using “loopreplace”, each field of the message in format of “global template” can be modified to be anomalous message so test suite need define only a little data in format of “global template”.
“loopreplace” can be used in “function” or “test case” in TTCN-3 language. We define syntax BNF of “LoopReplaceStatement” as follows:

\[
\text{FunctionStatement ::= ConfigurationStatements} \mid \text{TimerStatements} \mid \text{CommunicationStatements} \mid \text{BasicStatements} \mid \text{BehaviourStatements} \mid \text{ValidateStatements} \mid \text{SUTStatements} \mid \text{LoopReplaceStatement}
\]

\[
\text{LoopReplaceStatement ::= LoopReplaceKeyword} \text{ LoopReplacePar} \text{ LoopReplaceBody}
\]

\[
\text{LoopReplaceKeyword::= "loopreplace"}
\]

\[
\text{LoopRepalcePar ::= TemplateIdentifier} \text{ ExtendedFieldReference ConstIdentifier}
\]

\[
\text{LoopReplaceBody::=BeginChar FunctionInstance} \text{ SemiColon} \text{ EndChar}
\]

We give an example of statement using “loopreplace” as follows:

```
loopreplace HL1.Opt Boundary8bit_value {
  Onebyone_HL1();
}
```

The function of above two statements is to continually get the element of Boundary8bit_value, replace the “Opt” field of “HL1” and execute the function called “Onebyone_HL1()” which can inject anomalous message.

If we do not extend TTCN-3, the above two sentences will be described more complex. We also give a whole description of a compound anomalous test case:

```
testcase Onebyone_HL1_Opt() 
  runs on MyTestComponentAsync
  system SystemComponent {
    map(mtc:MyPortAsync, system:SystemPort1);
P1();
  }
```

In this test case, P1( ) leads neighbor state machine to “2-way”, then continually execute function “Onebyone_HL1( )”, i.e. inject anomalous Hello message, then transit the state to “Init” by force using the Event “1-way” and lead state to “2-way” again to inject next anomalous Hello message… After several loops, state verification is executed.

This test case belongs to Compound Anomalous Test Case-2. We can also describe Compound Anomalous Test Case-1 with “loopreplace”.

Based on BNF extension, we make use of Antlr [19] and StringTemplate [20] to further extend PITSv3. PITSv3 can translate the test suite described in TTCN-3 to Java code and generate ETS (Executable Test Suite) to carry out test practice. Antlr can identify the extended lexical sentence in the test suite using TTCN-3, make syntax analysis and build an out-parse tree. The semantics extension of TTCN-3 is based on this tree. We make use of StringTemplate to read this tree and generate source code in format of Java.

5. Test Suite and Test Practice of OSPFv2

OSPFv2 [17] is one of the most important and complex intra-domain routing protocol. There have been reports of vulnerabilities of OSPFv2 [18]. We test OSPFv2 using our method and results are given.

5.1. Test suite

According to OSPFv2 specification, there are five kinds of messages including: Hello, Database Description (DDP), Link State Request (LSR), Link State Update (LSU) and Link State Acknowledgment (Ack). OSPFv2 also defines five kinds of Link State Advertisements (LSAs). In addition, interface and neighbor state machine are defined in OSPFv2.

In test practice, we implement 120 compound anomalous test cases and each test case can inject about 5 anomalous messages. Test suite and test groups are given in Table 1.

<table>
<thead>
<tr>
<th>Test Group</th>
<th>Test Content (state / anomalous PDU received)</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSPF Head</td>
<td>Init / Hello</td>
<td>14</td>
</tr>
<tr>
<td>Hello</td>
<td>2-way / Hello</td>
<td>16</td>
</tr>
<tr>
<td>DDP0</td>
<td>Exstart / DDP0(without LSA Header)</td>
<td>8</td>
</tr>
<tr>
<td>DDP1</td>
<td>Exchange / DDP1(include one LSA Header)</td>
<td>22</td>
</tr>
<tr>
<td>LSR</td>
<td>Exchange / LSR</td>
<td>6</td>
</tr>
<tr>
<td>Ack</td>
<td>Exchange / Ack</td>
<td>16</td>
</tr>
<tr>
<td>LSA HEAD</td>
<td>Exchange / LSU(include Router LSA)</td>
<td>18</td>
</tr>
<tr>
<td>LSG_LRSA</td>
<td>Exchange / LSU(include Router LSA)</td>
<td>16</td>
</tr>
<tr>
<td>LSU Full</td>
<td>Full / LSU (include Network LSA)</td>
<td>4</td>
</tr>
</tbody>
</table>

5.2. Test practice and result analysis

Figure 7 shows the test scenario. The tester is called PITSv3 base on extended TTCN-3. We choose Zebra-0.94 [21] installed in Linux using Redhat 2.24 as IUT and carry out test practice using test suite in Table 1. Test results are shown in Table 2.
them for testing. Robustness testing on the level of we will mutate multiple fields of one message and use research on multi-field mutation testing. In future work, are also related to each other, it is necessary to carry out other various kinds of protocols. Platform based on extend TTCN-3 can also be used to test test requirements.

The work and contribution of this paper are given as follows: we build a formal model for mutation testing of protocol messages and propose our test method based on this model; the two kinds of Normal-Verification Sequence that we propose make the test verdict more reasonable; the compound anomalous test cases which we propose can simplify test sequences greatly; we apply semantic is another aspect we will pay attention to. We will also improve our method of TTCN-3 extension to automatically generate anomalous field values.

6. Conclusion and Outlook

Injecting lots of anomalous messages can effectively test the reliability, fault-tolerance and robustness of IUT. The work and contribution of this paper are given as follows: we build a formal model for mutation testing of protocol messages and propose our test method based on this model; the two kinds of Normal-Verification Sequence that we propose make the test verdict more reasonable; the compound anomalous test cases which we propose can simplify test sequences greatly; we apply TTCN-3 to mutation testing and extend it according to test requirements.

Using our test method, we test OSPFv2 implementation sufficiently. The results show that our test method has good capability of error-finding. Our test platform based on extend TTCN-3 can also be used to test other various kinds of protocols.

As protocol implementations may not parse the fields of receiving message in sequence and some field values are also related to each other, it is necessary to carry out research on multi-field mutation testing. In future work, we will mutate multiple fields of one message and use them for testing. Robustness testing on the level of semantic is another aspect we will pay attention to. We will also improve our method of TTCN-3 extension to automatically generate anomalous field values.

7. References