Framework and Model for Automated Interoperability Test and Its Application to ROHC

Sarolta Dibuz and Péter Krémer

Ericsson Telecommunications Hungary
P.O. Box 107, H-1300, Budapest 3, Hungary
{Sarolta.Dibuz,Peter.Kremer}@eth.ericsson.se

Abstract. In IP world, interoperability testing is heavily used to check the correctness of different implementations. Internet protocols have growing importance in communicating systems. In our paper we show an automatic interoperability test approach and also present its application on an IP-based protocol, ROHC. The primary goal of our work was to define an interoperability testing framework in TTCN-3 that can be used in general. After giving an overview on the ROHC protocol, we also describe the way we have conformance tested it. Then we present MAIT (Model for Automated Interoperability Test) and give detailed explanation on its components and their roles. At the end, we compare the advantages and disadvantages of conventional conformance testing and our interoperability testing model.

1 Introduction

OSI Conformance Testing Methodology [1] can be applied to Internet Protocols, as it has been shown in several papers ([8], [9], [10]). Nevertheless it is not spread in the Internet community. Interoperability testing checks if two different implementations of the same protocol have the capability of inter-working. It is used for testing prototypes built on RFCs and products implementing Internet protocol standards. Interoperability testing is such a well accepted verification method in IETF that a protocol draft can be an IETF standard only if there exists at least two inter-operating implementations for it.

In the telecom world conformance testing is more applied. ETSI, ITU, 3GPP and other standardization bodies develop conformance test suites. Vendors of telecom equipments or operators – the customers – are used to apply these conformance test suites to show conformance of the products or for type approval. Interoperability test is also done after the conformance tests. Its main function is to check if a new network element can inter-operate with the other nodes of the network on the main operation level.

Why is interoperability testing necessary if conformance testing is done or vice-versa? If two IUTs passes the same conformance test suite they can very likely inter-operate, as well. That is how conformance test is defined. But conformance test suites can not cover 100% of the protocol's functionalities. It may

D. Hogrefe and A. Wiles (Eds.): TestCom 2003, LNCS 2644, pp. 243–257, 2003.

happen that interoperability test of two IUTs fails even if the IUTs passed the conformance test. Especially, if the conformance test suite is not a well proven one. So it may happen if the conformance test suite is not a well proven one, that interoperability test of two IUTs fails even if the IUTs passed the conformance test. Protocol definitions contain optional elements. It may happen that the two IUTs implement or don't implement the optional features in a way that in certain cases this leads to problems in interoperability.

If two IUTs can inter-operate using a protocol it also may happen that one of them or both fail on a conformance test. There can be erroneous situations that were not tested during interoperability testing in which the IUTs fail. Erroneous situations can be triggered by conformance test but usually not with interoperability test. Another possibility for non-conforming IUTs, which can interoperate with each other is that the implementors misunderstood the protocol standard in the same way. So, the two IUTs can inter-operate but conformance test fails for both of them. In this case it is very likely that interoperability test with other IUTs would also fail.

Among others, IETF, ETSI, Sun and the TAHI project used to organize interoperability test events where developers can get together and perform interoperability tests. We have participated on several such events (like *Connectathon*, *IPv6 Bake-off* or *ROHC interop tests*) during the last 3 years. Our experience shows that every developer configures and starts his implementation manually. Moreover, at the end of each interoperability test case the analysis of the logs are also made by hand, which also implies that only the developer of the implementation can perform the interoperability test. We have seen a high need for a method and a tool that can help to automate the testing process.

The aim of our work was to give a framework that automates the testing process by using some parts of CTMF [1] and the flexibility of TTCN-3 [2]. This paper presents a model for interoperability test of Internet Protocols, and also gives an example of its application on ROHC protocol. We also compared the advantages and disadvantages of conventinal conformance test and this model of interoperability test in Section 6.

2 Related Work

Conformance testing methodology is well defined and has already proven its stability over the years. In contrary, interoperability testing has no such theoretical base. Several papers have been published related to interoperability testing but most of them deal with test suite generation or derivation [3], [4]. Others [5] perform monitoring and analyzing only, without triggering the IUTs for different actions or testing them actively.

[6] takes into consideration the interoperability test architecture, as well. It proposes three alternatives as possible architectures for interoperability tests. Unfortunately, this article doesn't address practical questions, which become extremely important when somebody wants to perform interoperability tests. For example, even the simplest architecture contains two Testers but the cooperation and synchronization of these Testers are not presented.

Another important issue is the communication between Testers and IUTs. These papers assume that interoperability test can be performed using the same service provider all the way. That is, the Testers send and receive protocol (the same protocol that is under test) messages through their standardized interfaces. Although it can be applied for several protocols but not applicable in general. In case of ROHC, three different interfaces are needed to perform interoperability tests. One is used to provide the necessary input. The second helps to configure the implementation, to trigger the tests and to check if the required action was taken. The third one monitors the exchanged packets. Telecom protocols are very well defined but the same can't be said for Internet Protocols. They usually leave a lot of questions open and the implementations have to make certain decisions (e.g. when to perform a mode change in ROHC). These standards doesn't specify upper and lower interfaces, thus it is not possible to construct an appropriate Upper Tester for these protocols.

Our approach differs in several ways. It gives not only a test architecture but also defines test components, their roles and the communication between them. It doesn't require a standardized upper tester, which is impossible to produce for Internet Protocols in most of the cases. This method also eliminates the problem of testing the states that are not observable from outside. On the other hand, our paper doesn't deal with automatic test suite generation at all.

3 Overview of ROHC

Nowadays, it seems reasonable that IP technology will be the most popular transfer mechanism. Moreover, Voice-over-IP solutions are getting improved and become more and more important. It is possible that mobile networks and cellular phones will also use IP technology in the future. In these systems, the efficient usage of radio bandwidth is crucial in order to serve as much subscribers as possible and to provide acceptable quality at the same time. The main problem with IP when used over wireless links is the large overhead of lower layer protocols. Assuming that RTP is used to transmit speech data, it requires 40 bytes in case of IPv4 and 60 bytes in case of IPv6. (RTP packets are embedded in a UDP packet, which is then carried by an IP packet.) Thus, the size of the payload can be as low as 15-20 octets, depending on the applied speech coder and frame sizes.

Reducing the headers' size is inevitable to improve efficiency but the existing methods ([11], [12]) don't perform well over wireless links ([13]). Basically, radio channels have the following shortcomings:

- high bit error rate ($10^{-3} \dots 10^{-2}$),
- long round trip time, high latency,
- low bandwidth.

An appropriate header compression scheme must be able to cope with these drawbacks as well as packet loss before the compression point. Since bandwidth is the most expensive resource, the compression ratio and the robustness have far

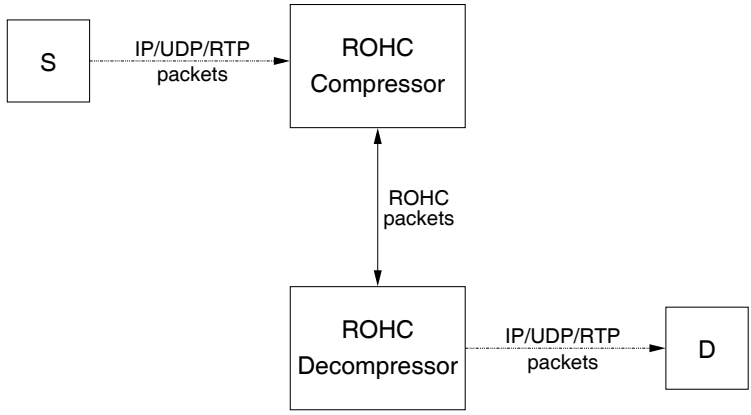

Fig. 1. A typical ROHC configuration

higher importance than the required processing capacity. It is important to note that the redundancy between header field values within a packet and especially between consecutive packets makes header compression possible.

ROHC (RObust Header Compression, [7]) is an IP header compression mechanism designed to perform well over wireless links. It classifies the header fields based on their behavior regarding the correlation to other fields. The conclusion is that most of the header fields never or seldom change. UDP Checksum (if enabled) and RTP Sequence Number (SN) have to be transferred directly, while the other three can be described as a function of SN and some other parameters. That is, in ideal case only the value of two fields (or only one if UDP Checksum is not enabled) have to be communicated explicitly. If a parameter of a function from SN to another field changes (e.g. irregularity in the input RTP stream), additional information is provided to update the necessary parameters of that function.

Figure 1 presents a typical ROHC configuration. The role of the nodes are the following:

- **Source Node (S)**: it produces the input packet stream, which is sent through the Compressor,
- **ROHC Compressor**: it compresses the incoming packets according to its current profile, mode and state, and transmits the resulting ROHC packets,
- **ROHC Decompressor**: restores the original packets (or something similar to the original) and forwards them to the Destination Node,
- **Destination Node (D)**: it is usually a regular application that decodes the original information from the output packet stream.

In ROHC, both the Compressor and the Decompressor has three states that determine the level of compression. States of the Compressor are: Initialization and Refresh (IR), First Order (FO) and Second Order (SO). In the beginning, the Compressor always starts in the lowest compression state (IR) and transits gradually to higher states if it "is sufficiently confident that the decompressor

has the information necessary to decompress a header compressed according to that state." [7]. The Compressor can decide the necessary compression state based on the variations in packet headers, on feedback from the Decompressor (positive or negative) or on timeout events. The Decompressor also starts in its lowest compression state (No Context – NC) and transits gradually to higher states: Static Context (SC) and Full Context (FC). Normally, the decompressor never leaves the "Full Context" state once it has reached it. Fallback to lower level states can occur on repeating decompression failures (due to packet loss or bit errors).

Besides states, which determines the level of compression there are also different modes of operation. They control the logic of state transitions and what actions to perform in each state. The current mode of operation can be changed if the Decompressor indicates a mode change request using a Feedback channel. ROHC uses the following modes:

- **Unidirectional (U)**: Packets can be sent in one direction only (from Compressor to Decompressor). Transitions between compressor states are performed only because of timeout events or irregularities in the input stream. Every Compressor and Decompressor start in this mode, mode transitions can be performed if a Feedback channel is available.
- **Bidirectional Optimistic (O)**: The difference to U-mode is that a Feedback channel (from Decompressor to Compressor) is used to send error recovery requests and optional acknowledgments of significant context updates. O-mode aims to maximize compression efficiency with the rare usage of Feedback channel.
- **Bidirectional Reliable (R)**: This mode uses the Feedback channel more intensively and has a stricter logic that ensures more robust context synchronization. R-mode aims to maximize robustness against loss and damage propagation even at high residual bit error rates.

The optimal mode cannot be selected without knowing the characteristics of the environment. The selection depends on the feedback abilities, error probabilities and distributions, etc.

The ROHC RFC defines four different profiles, which can compress different kinds of IP packets. These packet types are (numbers in parentheses denote the number of the profiles): uncompressed packets (0x0000), IP/UDP/RTP packets (0x0001), IP/UDP packets (0x0002), IP/ESP packets (0x0003). In this paper we deal only with profile 0x0001, the same method can be applied to other profiles, as well.

4 Conformance Test of ROHC

In this section we present the test configurations that we have used for the conformance test of ROHC in order to compare it with the interoperability test configuration. A typical configuration (Figure 1) consists of two ROHC implementations: a Compressor and a Decompressor. Since these nodes are working

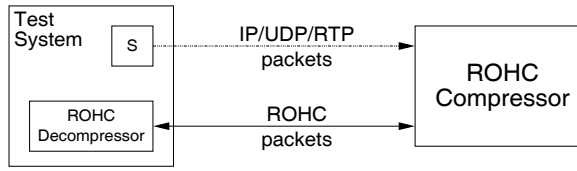

Fig. 2. Conformance test configuration for ROHC Compressor tests

separately and they are connected through a standardized interface, it is possible to test them separately, as well. We have used two different configurations (and test suites) to test Compressors and Decompressors. They are presented in the following two subsections.

4.1 Compressor Tests

To check a particular functionality in the Compressor the writer of the test suite must know how to compress certain RTP packets and how to decompress ROHC packets. This process is defined in the protocol standard, which can be very complicated in some cases. It is a nature of ROHC (and Internet Protocols in general) that an implementation can choose from a high number of possible actions in a certain state. In most of the cases, the RFC leaves it up to the implementation to decide which way to follow. Unfortunately, the same is true for the Decompressor, as well. These properties of the ROHC protocol make its conformance testing more difficult.

The test configuration for Compressor tests can be seen in Figure 2. The configuration consists of a *Test System* and a ROHC Compressor. The *Test System* emulates two nodes:

- **Source Node**, which generates the input IP/UDP/RTP stream for the Compressor
- **ROHC Decompressor**, which can process the compressed ROHC packets.

The two emulated nodes are using separate interfaces. Source Node needs an IP interface in order to send RTP packets. The RTP stream sent by the *Test System* is similar to that of an ordinary application would generate. The emulated Decompressor has a different kind of interface, since it needs to send and receive ROHC packets. Nowadays, two methods are used to transmit ROHC packets: ROHC-over-PPP and ROHC-over-UDP. Our *Test System* supports both kinds of transmission techniques. It is the nature of the protocol that the compressor can't generate a ROHC packet until it receives an RTP packet. Thus, we need to send an RTP packet through the IP interface, first. Then it will be compressed and the *Test System* will receive it through its ROHC interface. All of our tests follow this method and since the protocol assumes this sequential behavior, we didn't have to use parallel test components.

In some cases it is necessary to send a ROHC Feedback packet to the Compressor. The *Test System* uses its ROHC interface to send such packets. For example, to initiate a mode change the Decompressor has to send an appropriate Feedback packet to the Compressor.

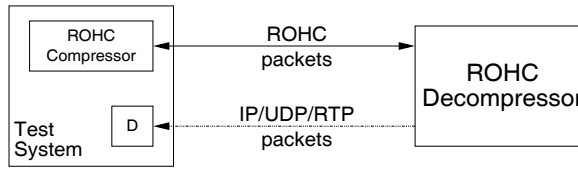

Fig. 3. Test configuration for ROHC Decompressor tests

Our test suite for ROHC Compressor tests consists of 100 TTCN-3 test cases and can check the following functions: mode transitions, context downgrade, changes in the incoming RTP stream, etc.

4.2 Decompressor Tests

Figure 3 shows the conformance test configuration for Decompressor tests. The main difference to the Compressor tests is that in this case the *Test System* emulates a ROHC Compressor and a Destination Node. The *Test System* sends ROHC packets to the Decompressor and processes the reconstructed RTP packets.

Testing a ROHC Decompressor is a bit more difficult than the case of a Compressor. The RFC doesn't define an interface that can be used to remote control a Decompressor. The lack of such an interface prevents us to install an upper tester application. Thus, it is impossible to test mode transitions automatically because the tester can't initiate a mode change remotely.

The test suite for Decompressors contains about 100 TTCN-3 test cases and verifies the correctness of mode transition, context downgrade, irregular changes in the original RTP stream. Unfortunately, these tests can't be executed automatically because of the missing upper tester.

5 Model for Automated Interoperability Test

The Model for Automated Interoperability Test (MAIT) that we have constructed is based on the experience we have gained in interoperability testing of Internet protocols (IPv6, Mobile IPv6, OSPF and ROHC). MAIT uses TTCN-3 [2] as a standardized test description language. This model gives a framework for automated interoperability test suites.

Figure 4 shows the architecture of MAIT. The configuration in this example consists of *IUT 1*, *IUT 2* and the *Test System*. *IUT 1* and *IUT 2* denote the two implementations, which are under test. The *Test System* handles the following tasks:

- remote controlling of *IUT 1*(*PTC R1*),
- remote controlling of *IUT 2*(*PTC R2*),
- monitoring the network (*PTC M*),
- sending protocol messages to IUTs (*PTC P*).

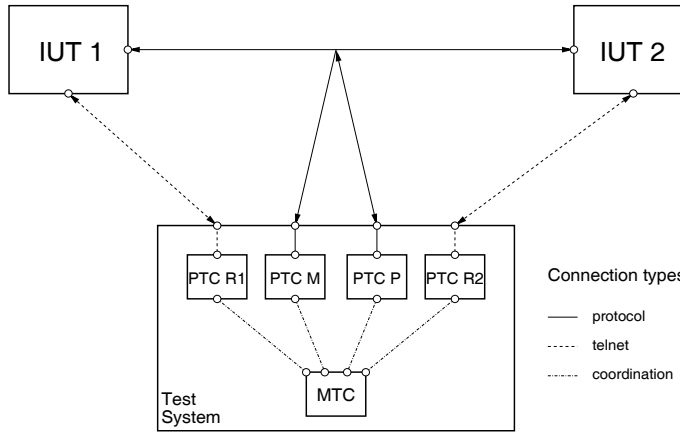

Fig. 4. Sample configuration of interop test

Since these tasks are independent from each other, they are implemented in Parallel Test Components (PTC). The Main Test Component (*MTC*) is used to coordinate the behavior of PTCs.

PTC R1 establishes a telnet connection between *IUT 1* and the *Test System*, this connection is then used to remote control *IUT 1*. The test component emulates an ordinary user: configures the IUT, starts an application or triggers a special test by giving the appropriate input. These kind of inputs can't be given through the IUT's standardized interfaces because most of the protocol standards don't specify the upper interface. Basically, it configures, starts and makes everything that only an experienced developer of that particular implementation can do.

PTC R2 creates the same type of connection but its tasks are slightly different. It also has to start and configure the implementation but then it checks if the test ran correctly and the necessary changes were made. It emulates the user of the other implementation, who follows the tests at the console and sees if everything is working correctly or not. It can be the establishment of a connection, a new record in a database or something else which shows that the test was successful. The messages that are sent on this type of connection are highly depend on the implementations. In order to avoid the re-compilation of the test suite, if a new implementation is tested, these messages are given as test suite parameters.

PTC P can send protocol messages to the implementations. There are cases when only a protocol message can trigger a certain action and it also gives the possibility to test inopportune behavior. For example, we can send a mode change request to *IUT 1*, which looks like as if it were sent by *IUT 2*. In this case, *IUT 1* follows the rules of mode transition and sends a packet according to its new mode. This packet will confuse *IUT 2* and gives the possibility to test inopportune behavior.

Every tester likes to know what happens on the wire, thus they always monitor the network that connects them to the other implementation. In general,

"tcpdump" (or a similar tool, e.g. ethereal) is used to record the packets. Synchronization of starting and stopping the packet recorder tool and the analysis of the log files are made manually in most of the cases. We have defined a separate test component in our model to handle and to automate these tasks. *PTC M* monitors the network, records and analyzes every protocol message that the implementations or *PTC P* send. We have separated the latter two test components because their tasks are slightly different. *PTC M* only receives and analyzes packets, while *PTC P* sends protocol messages (but it can receive, as well).

The function of *MTC* is to synchronize and to coordinate the behavior of PTCs. Similarly to conformance testing, the execution of a test case is divided into 3 phases (the names of the involved PTCs are shown in parentheses):

1. Configure the implementations for the test case (*PTC R1*, *PTC R2*).
2. Execute the test case (*PTC R1*, *PTC R2*, *PTC P*, *PTC M*).
3. Restore the original configuration (*PTC R1*, *PTC R2*).

The *MTC* creates all the PTCs, which start the configuration phase immediately. It then waits until all the PTCs finish the first phase (the PTCs send a "Ph_1End" signal to the *MTC* when they are ready with the first phase). In the next step, *MTC* starts the execution phase by sending a "Ph2_Start" signal to every PTC. After the execution of the second phase every PTC send a "Ph2_End" signal to the *MTC*. The signaling in third phase is similar to the second one. The synchronization model can be easily extended with additional phases, if needed. If an error occurs in a PTC at any phase, it sends a "Phn_Error" signal to the MTC (where n denotes the phase number), which stops all the running PTCs and the test ends immediately. Figure 5 shows an example of the communication between *MTC*, *PTC R1* and *IUT 1*.

5.1 Modifications to ROHC

To perform interoperability test of ROHC one needs a source and a destination node that can generate and receive RTP packets. In real life this RTP stream is generated by an application but in case of testing we need a separate, flexible and supervised RTP packet generator. Only such a generator can ensure that the input RTP stream is syntactically and semantically correct and has controlled behavior. The packet generator must be able to generate streams with regular and irregular behavior, as well. Irregularity can be a jump in a monotonically increasing value or simply a packet loss. A destination node is also required, which must be able to check if the output RTP stream is syntactically and semantically correct, and to compare input and output streams. Thus, both nodes must be able to record RTP packets, destination node must be able to read the records of the source node and it must be also able to check whether the two streams are equivalent or not.

In order to provide a solution that better suits the need of ROHC, we have modified the original model. We have added two more test components (*PTC S*,

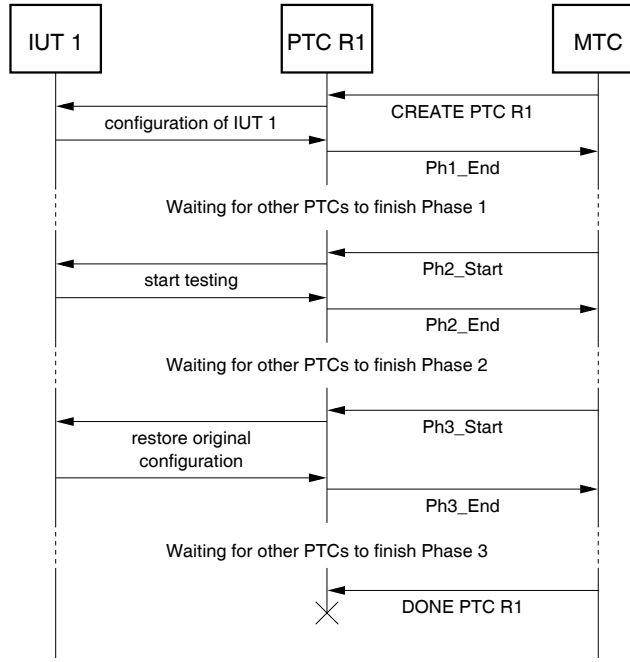

Fig. 5. Synchronization of *PTC R1* and *MTC*

PTC D) that are acting as a source and a destination node. The extended model for ROHC can be seen in Figure 6.

PTC S generates the input RTP stream for the Compressor (just like a packet generator would do) and *PTC D* receives the output RTP stream from the Decompressor. *PTC S* and *PTC D* can be configured on a test case basis and *PTC D* exactly knows the content of each packet that *PTC S* sent. The comparison of the input and the output streams became quite trivial in this case. This method eliminates the necessity of recording the packets on both sides and also the need for manual checking of the files. Thus, the whole testing process can be automated and fully controlled by using this method.

6 Advantages and Disadvantages of MAIT

In this section we describe the advantages and disadvantages of MAIT compared to ordinary conformance testing. First, let's consider the process of preparing a test suite. To write a conformance test suite, a stable description of the protocol is needed. In this sense, stable means that it doesn't change frequently, the changes doesn't affect the elementary parts of the protocol (i.e. connection establishment method or new fields in a message), etc. Then comes the development of the test purposes and the test cases. Writing test purposes usually requires the creation of some kind of a state machine with appropriate inputs, outputs and transitions.

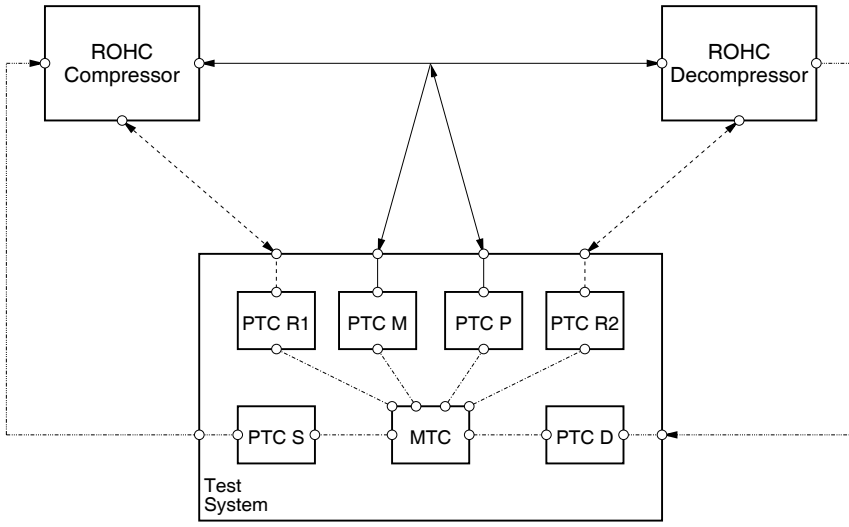

Fig. 6. Extended model of interoperability test

Test cases can be produced for the states that has definite behavior, clearly defined and reachable with the available input sequences. The exact definition of the messages (and the value of their parameters) that the test cases are sending and receiving is one of the most difficult parts of test suite writing. Finally, the test system needs to be adapted to the protocol, as well. Once the conformance test suite is ready, it can be executed against any implementation.

Naturally, a protocol description is also required in case of interoperability test, but it doesn't have to be so mature. The changes of the protocol doesn't necessarily cause trouble during the tests, because both parties' behavior is updated (since we are testing the interoperability of two different implementations of the same protocol). Let's change the packet formats in ROHC! We don't have to alter the input stream, the task of the protocol haven't changed, thus the output stream must be the same, as well. Moreover, we doesn't have to change the configuration commands. The only things that has to be updated are the type and template definitions of the protocol. It only affects *PTC M* and *PTC P* processes. To write test purposes one has to take into consideration all the possible scenarios and input sequences, just like in conformance test. Instead of creating a state machine, only the input sequences are needed in this case. This input sequence differs from the one used in conformance test. For ROHC it is a set of RTP streams, where each stream differs from the others and invokes a different action of the IUT. In other cases it can be a set of commands that establishes a connection but it is important to note that this input sequence doesn't consist of protocol messages. This nature of interoperability test make the writing of a test suite much more easier and faster.

Each test case represents a different interoperability scenario. These scenarios concentrate on the most important functionalities of the protocol. This means

that the interoperability test suite is not covering all aspects of a protocol as a conformance test suite. Since MAIT defines the structure of the test cases and the implementation specific commands can be set as parameter, it provides easy-to-understand and well-structured tests. These tests describe the protocol functionalities in a clearer way than the text of the RFC. With the interoperability test the logics of the protocol is also verified.

6.1 Test Execution

If we take a look at the execution of tests, we see that Conformance testing is executed automatically. If the test suite doesn't change between two executions (which is the normal case) it produces repeatable test results. Based on these results, we can compare two implementations regarding their functionalities. Ordinary interoperability testing is usually done manually. Therefore it lacks the ability to repeat the same tests and to compare implementations based on the test results. Having the MAIT model used these disadvantages can be avoided. MAIT uses predefined test cases in the same way as conformance test uses. It also gives the possibility to automatically execute interoperability test cases and eliminates the need for human interaction (e.g. finding out if the test was successful or not, bug tracking, etc.). Since the test cases are defined in a test suite, the tests become repeatable and the results reproducible and comparable, as well.

The number of performed conformance tests grow linearly with the number of the implementations, as a test suite have to be executed on an implementation once. The reason is that a conformance test suite can also be seen as a reference implementation, by definition. In case of interoperability test, the number of test executions grow quadratically with the number of implementations because we don't have such a reference. However, we don't have a reference implementation by definition, we can appoint one or more implementations as reference if they fulfill certain requirements[1]. One requirement can be the successful interoperability test with a given number of other implementations. Others can be set easily upon common agreement of the implementors. The number of test executions decreases drastically if appointed reference implementations exist.

6.2 The Role of Testing in Protocol Design

Both conformance and interoperability testing is used to prove that products fulfill the requirements stated in the standard. Conformance testing is used usually when we have the protocol definition available as a standard. Hence, conformance testing is not part of protocol design, when the protocol itself is created but it is heavily used if the standard has a mature version and numerous implementations are available on the market.

[1] The authors don't intend to solve the problem in this paper. They just recommend some possible solution without giving further explanations, arguments or proofs.

In IETF the process of protocol design is iterative in this sense. Vendors implement prototypes of the protocol written in the RFC. The protocol is designed further in the IETF working-group based on the experiments of the prototyping. It is usual to find errors and ambiguities in the protocol and in its specification during interoperability test. They will be corrected, clarified in the next version of the specification or added as a separate document [14]. Either way, the specification evolves as interoperability tests are executed more and more times. It is easier to modify the interoperability test suite in case the protocol is modified, as it is smaller in size and it has a stronger dependency on the inputs than on the protocol's logic and messages. Thus, we can say that interoperability testing is an integral part of the protocol design.

6.3 Implementation Dependency and Reuse of Tests

Conformance test is implementation independent, since it uses only the standardized interfaces of implementations. This attribute makes it possible to use the same test suite for different implementations. Interoperability tests are usually performed for Internet Protocols and these protocols don't have such a detailed specification. It implies that it cannot be guaranteed that these tests will be implementation independent. We have shown in case of ROHC that implementation specific messages are used to configure the implementations, to trigger tests and to check if the required action was taken. This dependency is inevitable if we want to check all the possible interoperability test scenarios. In case of using MAIT, these implementation dependent messages can be given as parameters, so it is not necessary to recompile the Executable Test Suite. Recompilation may need tools and equipments that are usually not available at the site where interoperability testing takes place.

Nowadays, TTCN is a common language for writing conformance test suites. Its latest version [2] is general and flexible enough to describe interoperability test suites, as well. If the two different types of test suites are written in the same language (and MAIT is written in TTCN-3) then some parts of it can be reused. This situation becomes more and more likely as TTCN-3 based test tools get used widespread. The interoperability test is part of the protocol design (at least in IETF), protocol specification, implementations and tests are developed together. It also means that an interoperability test suite is written before the conformance test suite and both use the same description language. That is, results, experiences and parts of the TTCN-3 source code of the interoperability tests can be reused in conformance tests. These parts are the definitions of the protocol's data types and messages, templates and timers.

The test environment also have to be adapted to the particular protocol. This is mainly done by a software module that can transmit and transform protocol messages between the test tool and the underlying service provider. This adaptation module is reusable, too. Our experience shows that the reusable parts (data type and template definitions, adaptation module, etc.) takes 20-40% of the work needed to create an executable conformance test suite. Moreover, it is also possible to record packets during interoperability test and save their

contents in order to use in conformance testing. We have already shown that the content of the protocol messages is very important in conformance testing and creation of them is one of the most difficult tasks. Storing the packets for later use can also save reasonable amount of time. In other words, writing an executable conformance test suite could take as high as 50% less time.

7 Conclusion

This paper presented the MAIT model for automated interoperability test and also defined a framework that uses this model. We gave detailed explanation on the components of the model and the role of each component. We showed the advantages and disadvantages of this model and compared it to the properties of conformance testing.

Using our method it is much more easier and faster to derive an interoperability test suite than a conformance test suite from a textual description of a protocol. It is not necessary to create a state machine, to figure out the order and the content of messages to be able to write a test case. The writer of the test suite only has to know what the IUT should do (i.e. set up a connection) in that test case and ask for the configuration commands from the implementors. It is the implementor who knows every detail about the implementation and can give the necessary information. These commands are handled as test suite parameters, so the test suite writer doesn't have to know them at all.

Unlike in case of ordinary interoperability test, the MAIT model stores the interoperability test scenarios in a form of TTCN-3 test cases. Thus, it gives the possibility to rerun the same tests on the same implementation at a different time. It ensures the ability to repeat the same test and to compare the implementations based on the interoperability test results.

Although the model uses implementation specific configuration commands, it doesn't depend on the IUT or on the protocol. It is because these commands are passed as parameters (executable conformance test suites also use similar parameters). That is, a change in these commands doesn't require the change of the test suite. In other words, an implementation can alter in a way that affects its configuration commands but the MAIT test suite can remain the very same. It also means that testing an implementation of another vendor only needs the configuration commands of that implementation.

With the MAIT method it is easier to change the interoperability test cases if the protocol specification change. This situation happens quite often during in the design phase of a protocol. Waiting for a stable specification is not feasible in practice, because interoperability testing is also used to verify the protocol itself (IETF requires two independent inter-operating implementations to accept a protocol specification as a Proposed Standard).

Table 1 shows the needed testing types in different phases of a protocol. During protocol design phase, no stable specification is available, it can change day by day. So, it requires a testing type where it is possible to quickly adapt to changes. Protocol life-cycle phase presumes a mature standard and it gives the possibility to develop a product that is an implementation of that particular

Table 1. Testing needs in different phases of a protocol

Protocol design	prototype	Interoperability Test
Protocol life-cycle	product (protocol implemenetation)	Conformance Test

protocol. It can be seen that the need for an interoperability test suite precedes the need for a conformance test suite. But when the protocol definition reaches a stable state as a standard, conformance testing should be applied first to test the correctness and conformance of the products implementing the protocol. That is why it is important that parts of a MAIT-based interoperability test suite can be reused in a conformance test suite.

References

1. OSI - Open System Interconnection, Conformance testing methodology and framework, ISO/IEC 9646, 1997.
2. ETSI, The Testing and Test Control Notation version 3, ETSI ES 201 873, v2.2.0, May 2002.
3. S. Kang, J. Shin and M. Kim: Interoperability test suite derivation for communication protocols, The International Journal of Computer and Telecommunications Networking, Vol. 22, Num. 3, pp. 347-364, March 200.
4. N. Griffeth, R. Hao, D. Lee, R.K. Sinha: Integrated System Interoperability Testing with Applications to VoIP, Formal Methods for Distributed System Development (FORTE/PSTV 2000), Pisa, Italy, October 2000.
5. T. Kato, T. Ogishi, H. Shinbo, Y. Miyake, A. Idoue and K. Suzuki: Interoperability Testing System of TCP/IP Based Communication Systems in Operational Environment, Testing of Communicating Systems, Ottawa, Canada, September 2000.
6. J. Shin and S. Kang: Interoperability Test Suite Derivation for the ATM/B-ISDN Signaling Protocol, Testing of Communicating Systems, Tomsk, Russia, September 1998.
7. C. Bormann (ed.): RObust Header Compression (ROHC), RFC 3095, July 2001
8. R. Gecse: Conformance testing methodology of Internet protocols, Testing of Communicating Systems, Tomsk, Russia, September 1998.
9. R. Gecse, P. Krémer: Automated test of TCP congestion control algorithms, Testing of Communicating Systems, Budapest, Hungary, September 1999.
10. T. Csöndes, S. Dibuz, P. Krémer: Experiments on IPv6 testing, Testing of Communicating Systems, Ottawa, Canada, September 2000.
11. M. Degermark, B. Nordgren and S. Pink: IP Header Compression, RFC 2507, February 1999
12. S. Casner and V. Jacobson: Compressing IP/UDP/RTP Headers for Low-Speed Serial Links, RFC 2508, February 1999.
13. M. Degermark, H. Hannu, L.E. Jonsson, K. Svanbro: Evaluation of CRTP Performance over Cellular Radio Networks, IEEE Personal Communication Magazine, Volume 7, number 4, pp. 20-25, August 2000.
14. P. Krémer, L. E. Jonsson: Implementer's Guide, May 2002, http://standards.ericsson.net/kremer/draft-ietf-rohc-rtp-impguide-01.txt.