Using TTCN-3 in the Internet Community: an Experiment with the RIPng Protocol

Ariel Sabiguero[1,2], Anthony Baire[1], Annie Floch[1], and César Viho[1]

[1] IRISA
Campus de Beaulieu
35042 Rennes CEDEX, France
{asabigue,abaire,afloch,viho}@irisa.fr,
http://www.irisa.fr/armor
[2] Instituto de Computación, Facultad de Ingeniería, Universidad de la República
J. Herrera y Reissig 565, Montevideo, Uruguay
asabigue@fing.edu.uy
http://www.fing.edu.uy/inco

Abstract. This paper relates an experiment in using TTCN-3 for developing conformance test suite for the RIPng protocol. This paper identifies main issues that one may consider when trying to develop executable conformance tests using TTCN-3 on IP related protocols. Main issues that any new TTCN-3 user may deal with are highlighted. Provided solutions are presented together with main features that have to be included in TTCN-3 based tools to ease test development.

Keywords: Internet Protocol, TTCN-3, RIPng, Conformance testing

1 Introduction

TTCN-3 has been designed to provide a well suited language for any kind of testing activity [1–4], from abstract test suites specification to executable test suites [5, 6]. As it is a new language, there is not enough maturity regarding its usage and environments that are supposed to ease TTCN-3 usage. The European community, through the European Telecommunications Standards Institute (ETSI), promotes the use of the TTCN-3 language for testing purposes [7, 8].

In the Internet community in general, TTCN-3 is not widely adopted. As a newcomer into the IP testing arena, it is required to "provide more" or "do better" so as to justify the switch from one way of testing to another. Moreover, it is even unfavorably criticized. This is mainly due to the confusion with it's predecessor TTCN-2, which was considered a rigid language and difficult to generate tests for new protocols. This 'bad reputation' applies for testing the new protocols developed for the new version of the Internet Protocol, called IPv6. Indeed, most of the existing test suites are developed using IPv6 dedicated languages and tools. The most famous one is the v6eval toolbox (http://www.tahi.org)

developed by the Japanese TAHI project. In this context, it is difficult to convince people to use TTCN-3 without showing real executable test suites for at least a simple IPv6 protocol.

The objective of the present work is to gain experience using the TTCN-3 language and tools while addressing a pending IPv6 test conformance problem. The Routing Internet Protocol for IPv6 (RIPng [9]) has the advantage of being relatively simple (at least compared to other IPv6 related routing protocols), while an important and widely deployed protocol in small to medium organizations. This work also aims at proving to the Internet community that TTCN-3 is can be used for testing, covering all steps from abstract test suites (ATS) to executable test suites (ETS). It was also important to identify main issues when testing with TTCN-3 and providing solutions that may help simplifying future test generation.

The methodology behind this work was restricted in scope as the goal was to be able to obtain ETS to be executed against real implementations during the IPv6 interoperability event organized by the ETSI/Plugtests Service in October 2004. After October, we re-engineered some details and produced a new ETS that was executed also against real implementations, with a tight schedule for the TAHI IPv6 Interoperability event in January 2005. Thus, we followed a straightforward approach due to time constraints: some decisions were based on time-to-executable-test parameters. On the other hand, one may note that this kind of requirements also corresponds to the real Internet community and industry requirements of having ETS as soon as the need of testing is identified.

Amongst all available TTCN-3 tools, the choice was made for a tool that allowed us to have access to the source code if necessary. Indeed, due to the youngness of TTCN-3 and our current knowledge in using this new language, it was important to use a tool which allows libraries source code modification if needed. Work on portability of the ATS across different TTCN-3 tools has been started and is in progress now.

As a result of this work, a RIPng conformance ATS/ETS based on TTCN-3 is now available. These tests have been ran against real implementations during the last IPv6 ETSI-Plugtests interoperability event in October and during the Japanese IPv6 TAHI interoperability event in January 2005. Test results were considered of interest by participants. Doing this work and following the approach indicated above, we face several issues that any new TTCN-3 user may have to deal with. Amongst other results, these main issues have been highlighted, and our solutions are introduced. Some ideas that may help in easing test development using TTCN-3 are proposed.

The rest of this paper is organized as follows. Section 2 explains with more details the context of the work and provides a brief description of the RIPng protocol. Main TTCN-3 components that have to be developed are described. Section 3 outlines different steps to obtain TTCN-3 based test suites for the RIPng protocol. Problems encountered during test development phase and their solutions are also presented. Section 4 presents some results and lessons learned from this experiments in using TTCN-3 for RIPng testing. Some ideas that might

help in easing other similar effort are presented. Conclusions of this work can be found in Section 5, where future work is suggested.

2 Background of the experiment

We have been involved for years in developing IPv6 conformance tests suites. The tool used is `v6eval`, developed by TAHI project (`http://www.tahi.org/`). In such line of research, we are working now to produce test suites for several IPv6 routing protocols.

One important reason behind the present work for us is to find provider-independent tools and languages for defining test suites. TTCN-3 is presented as a modern standardized abstract language, test oriented and provider independent. Tool providers implement their solutions according to the standards, but independently. It is widely accepted that multi-provider scenarios lead to more complete and general languages and tools than single provider ones. The lack of free/open reference TTCN-3 implementations also presents some limitations to a community that has been working with open/free tools and operating systems.

Our primary motivation was to experiment with the ability of TTCN-3 for our testing purposes with real and concrete IPv6 protocol. On the other hand, we wanted to show to the IPv6 community that TTCN-3 can be used for this purpose. One way to prove that is to have executable test suites built with TTCN-3 language and tools, which can be used during interoperability sessions.

2.1 RIPng brief overview

RIPng[9] is the logical step of the well known IPv4 family of RIP protocols into IPv6 world. RIPng stands for *Routing Information Protocol - Next Generation*. RIP belongs to the class of algorithms known as "distance vector algorithms". Distance-vector algorithms are based on the exchange of only a small amount of information. Each network node that participates in the routing protocol must be a router as IPv6 protocol provides other mechanisms for router discovery, and it is assumed to keep information about all destinations within the system.

Limitations of RIP include network diameter restrictions, counting to infinity to resolve loop situations and the lack of metrics based on dynamic. Some of the limitations are not *per se* limitations, but they are a consequence of the design of the protocol. RIP is not intended to be used as Internet's single routing protocol, but as an Autonomous System (AS) internal protocol. RIPng is an UDP-based protocol and listens on the port 521. It is a message oriented protocol (implemented messages are 1-request and 2-response), based on distributed intelligence, without any distinguished node. The figure 1 shows a typical RIPng deployment scenario, where 6 interconnected routers exchange routing information as request-response messages.

IPv6 protocol defines and implements three different types of communication destinations, which are: unicast, anycast and multicast. This enhancements at network/transport layers provides better support for protocols using their

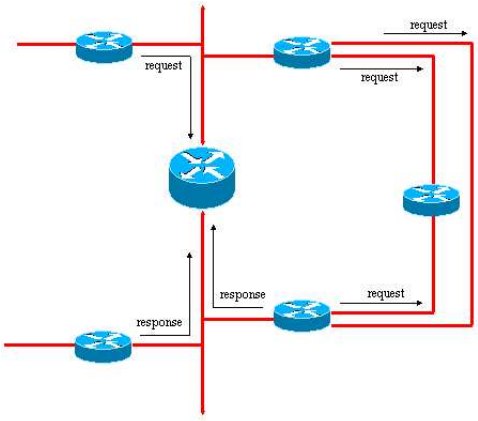

Fig. 1. Autonomous System RIPng messaging

services. RIPng uses both unicast and multicast mechanisms for inter router communication, according to the kind of message exchanged. The multicast address ff02::9 is reserved as the all-rip-routers group, which is used except in some non-multicast channels, where explicit network addresses have to be used.

Authentication mechanisms have better grounds on IPv6 protocol stack and thus, are removed from RIPng protocol itself.

2.2 TTCN-3 main components

TTCN-3 is a pretty new language (current TTCN-3 Core Language[10] was published on 02-2003) with only a first generation of compliers and tools supporting it. TTCN-3 was designed to be able to incorporate testing capabilities not present on other programming languages, and was also cleared from OSI peculiarities (that previous versions suffered). TTCN-3 is designed to be flexible enough to be applied to any kind of reactive system tests.

The layout of a TTCN-3 test system general structure is shown in figure 2. As usual, this test system is supposed to be executed against a system under test (SUT). Each block in the figure represents an entity implementing a particular aspect required by a test system. The test system user interacts with the Test Management (TM) and uses the general test execution management functionality. The TM entity is responsible for the global test management. The TTCN-3 Executable (TE) implements the functionality defined as TTCN-3 modules, which can be structured into sub-modules and import definitions from other modules. Modules have a definition part (defines test components, communication ports, data types, constants, test data templates, etc.) and a control part (which is responsible for calling test cases and controlling their execution). Other test layout dependent parameters are defined at the SUT Adapter (SA) and the Platform Adapter (PA). A TTCN-3 test system has two main internal

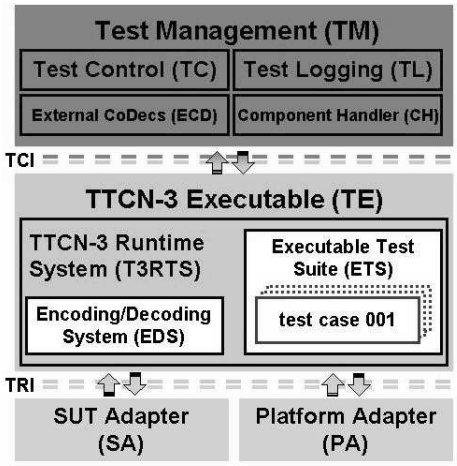

Fig. 2. TTCN-3 Test System Architecture

interfaces, the TTCN-3 Control Interface (TCI) and the TTCN-3 Runtime Interface (TRI). TCI specifies the interface between Test Management (TM) and TTCN-3 Executable (TE) entities. TRI interface specifies the interfaces between TE, SUT Adapter (SA) and Platform Adapter (PA) entities.

Figure 3 shows the modules and main methodological tasks that have to be developed to produce test suites. The blocks named `RIPng Test Cases` and `RIPng Templates` correspond to the tasks required to define the TTCN-3 Executable block on figure 2. The blocks named `SUT Parameters` and `PCO Definition` correspond to parameters required by the SA to interface with the SUT.

3 The experiments

We have a broad experience on the IPv6 field, while these experiments are our first practical approach to TTCN-3. Nevertheless, both our experience and the methodology used in the IPv6 community matches the principles suggested in [11]. The hands-on experiences with TTCN-3 presented in this paper tries to answer whether the language and methodology are ready for addressing the strong needs of the IPv6 test community. It is worth mentioning that the Internet community, for more than 20 years now, is a very pragmatic environment, who do not care about the way the tools are designed, but focus on the way they can quickly answer to their needs. Our goal was to develop tests for RIPng in a short time with existing new tools. This work documents field experience with TTCN-3, but does not intend to promote a methodology.

It is known that a black-box approach to conformance testing will only allow us to exchange signals with the System Under Test (SUT): in this case, signals are

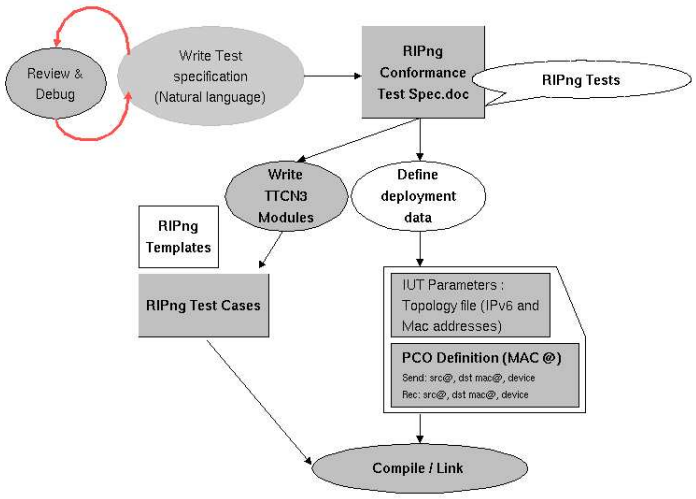

Fig. 3. TTCN-3 based initial approach of test specification

RIPng messages. Designed test cases consisted of exchanging routing information with the SUT and later sending IP probes to selected destinations so as to determine the way routing information is not only learned and shared by the SUT, but also, applied on its own routing decisions.

To be able to specify TTCN-3 test cases we had to obtain a tool and define the needed modules according to our test purposes. It was also required to provide the SUT Adapter (SA) with proper definitions so that the mapping between TTCN-3 components communication ports and test system interface ports is done. After this, the ETS is generated.

3.1 Approach for TTCN-3 test specification

Routing Table Entries (RTE) are the key elements exchanged within RIPng messages. Each router is supposed to have some sort of routing table with at least the following information: the IPv6 prefix of the destination, a metric, the IPv6 address of the next router along the path to that destination, a flag and various timers associated with the route. This suggests that basic routing operations to test shall be related to RTE maintenance like: RTE creation, RTE update, RTE deletion, RTE request.

Maybe the simplest test topology would consist of two routers and the SUT, each connected to a different physical interface of the SUT, but it will not permit to perform the desired probes. From the test purposes settled we decided to build a more complex network layout, shown on figure 4. The small box in the center represents the role that the SUT plays in the topology, while the rest of it, marked

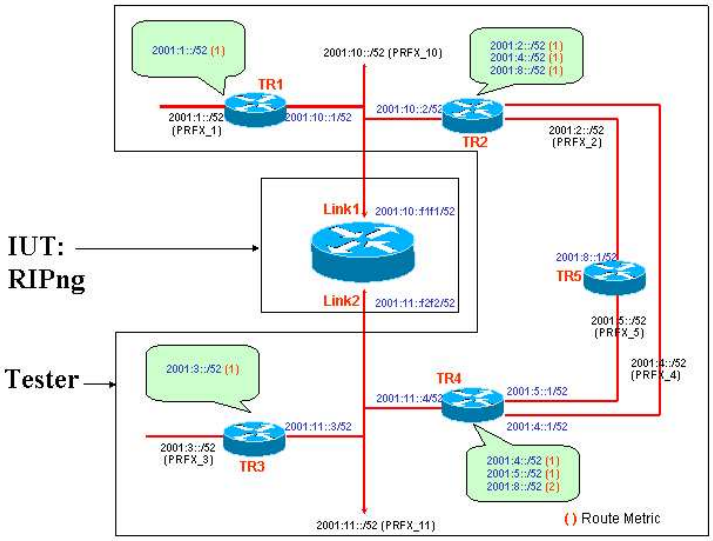

Fig. 4. RIPng testing topology

as `Tester` represents what has to be developed to perform the tests. For specific test purposes we selected -projected- the relevant routers that allow inspection of the desired property and specified the particular ATS only considering it. This methodology simplifies test design because we have a single well known network, and it allows us to concentrate on details of each test purpose by projection of relevant smaller parts of the network.

It was required right from the first test definitions to be able to emulate more than one router to explore even simple protocol behavior and properties. This fact makes us define and handle several Points of Control and Observation PCO. The distribution of PCO over single or multiple test execution threads or processes promotes the discussion between parallel *vs.* single party testing, or in other words, a Master Test Component (MTC) with Parallel Test Components (PTC) *vs.* single MTC. Protocol complexity was not an issue at this point, as the protocol itself is simple: both solutions are adequate for test requirements. From our previous experiences and the lack of time for enough testing of the TTCN-3 parallel possibilities and API, we decided for a solution with a single MTC that handles all required PCO. The decision of using a single node to emulate the whole network topology allows us to avoid all parallel synchronization problems. This decision also considers easy deployment and testing reusage: it is simpler to deploy a single device than a configuration with 6 nodes. We believe that naive deployment of PTC corresponding to each emulated router would have produced test suites with different characteristics. Complexity of test setup would have increased considerably as separate process on different machines had to be configured, etc.

Another important decision was the tool selection, which was done considering all the existing tools known to us (testing_tech, Telelogic, Danet, OpenTTCN, etc.). At the time of the selection all available tools were equally eligible as they all implemented TTCN-3 required components. Also, none of them provided already built IPv6 libraries that might have helped with the building blocks for RIPng tests. The decision was based on our experience testing with C++ tools and licensing conditions that allowed us not only to use the tool for academic purposes, but also to have access to the source code when needed. Other aspects considered were Integrated Development Environments (IDE) and tools provided that help with simple and repetitive tasks. From all those testing tools available the choice was made for Danet's testing tool (`http://www.danet.de`).

3.2 PCOs management

Points of Control and Observation (PCO) play a very important role on what can be observed out of a system. Proper selection of PCO placement would allow better and detailed protocol inspection. As RIPng is a UDP based protocol the first test design tried to place PCO at UDP level, as shown in the figure 5. In TTCN-3 PCO are referred as ports.

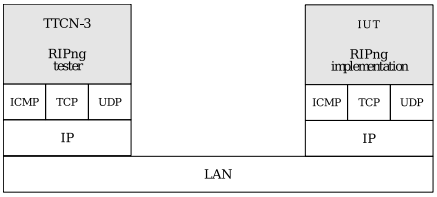

Fig. 5. RIPng testing architecture, UDP level PCO

It was not possible to code a single tester using TTCN-3 that was able to emulate several routers. The selected tool only implements two types of ports: serial and socket. Serial does not apply for Ethernet communication, and socket is implemented using underlying operating system protocol stack services at socket level, thus it is not possible to simulate traffic to and from different routers: it would be necessary to define different IPv6 addresses and Ethernet MAC addresses. TTCN-3 definition is independent of this low level details. Thus, it does not allow dynamic definition of MAC/IPv6 addresses associated to ports on every implementation.

Another observation is that we do not only need UDP services: ICMP echos are sent through the SUT so as to check the routing decisions at a certain moment.

The figure 6 shows all the parts -grayed- of the protocol stack that had to be addressed with the test. The main difficulty was that when more than one router

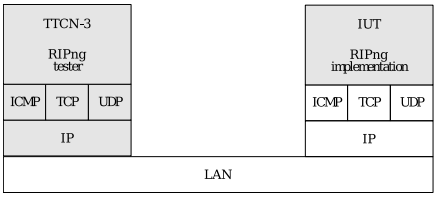

Fig. 6. RIPng testing architecture, link layer/IP level PCO

had to be emulated using a single MTC, the IPv6 native stack on the host had to be disabled and all the steps of the communication had to be emulated from TTCN-3 modules. It would have been also the same situation with several PTC running on the same machine. Several issues arose during the development phase. Both the TTCN-3 tool and language were not designed to handle this situation of multiple host emulation using a single Network Interface Card (NIC). Due to time constraints we worked out the problems by changing some aspects of the tool implementation by recoding parts of TTCN-3 primitives. We changed TRI basic primitives so as to handle link layer PCO, which were not implemented in Danet's tool. The modification consisted in adding a new type of port that handle Ethernet communication, but at the physical interface level. With the modifications introduced we were able to emulate as many hosts -form data link layer up- as required from a single real host.

This kind of handling increased the complexity of the ATS as not only RIPng protocol communications had to be implemented. Required UDP assembly and disassembly of packets also was needed, including checksum and packet length calculation. IPv6 layer assembly and disassembly of packets was also mandatory. In the end also data link layer parameter handling had to be introduced to transmit packets with the corresponding MAC address of the router emulated. Moreover, the reception of the packets and their corresponding processing had to be handled.

Other link maintenance aspects of IPv6 Neighbor Discovery[12] (ND) algorithm had to be addressed. IPv6 relies several host autoconfiguration tasks to the ND. Thus, for correct node emulation, ND signaling is necessary.

TTCN-3 template definition was not versatile enough to allow efficient matching of incoming data. The solution found was to create as many PCO as couples of communicating addresses required. Due to the way IPv6 handles addresses, each emulated node was associated to several addresses (unicast and multicast). To fulfill this multiple addressing scenario, several PCO were introduced. The complexity generated by this fact was significant, both at ATS coding and at tool modification level. ATS legibility was also an important issue as classification of messages received becomes complicated.

3.3 Coding/decoding, libraries, etc.

As stated before, the communication between RIPng nodes is message-oriented. Message definition has a low level of abstraction and coding/decoding is done dependent on the position of bits within the frame. Figure 7 presents RIPng packet as defined in the RFC 2080[9] with its corresponding IPv6 header prepended, without any IPv6 options.

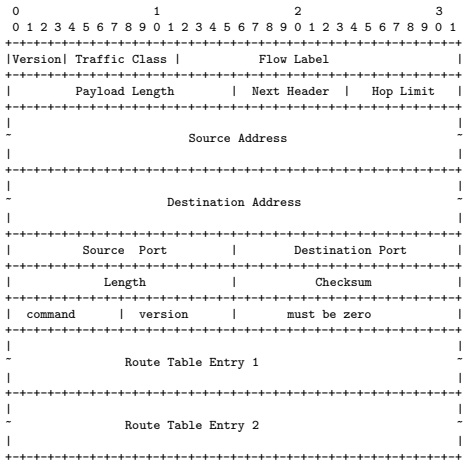

Fig. 7. RIPng packet format

Several codification issues needed to be resolved in order to define a TTCN-3 module that abstracts the RIPng packet. First of all, some fields are hard-coded always, like the protocol version, which is `'0110'B` for all IPv6 tests. Other fields are parameters of templates, like prefixes and prefix length values. Some fields are parameters of the component, like source and destination addresses (different from one tested router to other). Finally, others need to be calculated each time a packet is about to be transmitted, like payload length and checksum values.

We found that there is no easy mechanism, like the ones defined on the RFC 2373[13], for IPv6 address text representation. When defining parameters for a component, its IPv6 address `2001:2::1` had to be coded. In our environment, XML files are used (see figure 8).

To ease TTCN-3 based IPv6 test generation, a test environment shall provide standardized methods for network address handling and representation.

TTCN-3 data type definitions were coded to provide abstract description of IPv6 packets. Templates are built based on data type definitions. Figure 9 shows an example of a template defined.

```
<RUT_LINK2_GLOBAL_ADDRESS1  moduleId="IPv6RouterInterface\">
    <OctetStringValue valueKind="4\">
        20010002000000000000000000000001
    </OctetStringValue>
</RUT_LINK2_GLOBAL_ADDRESS1>
```

Fig. 8. Markup defining an IPv6 address

```
template IPv6PacketType RIPngRequestTable_tp
  (IPv6AddressType source, IPv6AddressType est,
   IPv6AddressType P1, UInt8 PF1, IPv6AddressType P2, UInt8 PF2) :=
{
Ipv6Header := { Version := int2bit (6, 4),
                TrafficClass := 0,
                FlowLabel := int2bit (0, 20),
                PayloadLength := 0, // CALCULATED BEFORE SENDING
                NextHeader := NextHeaderUDP,
                HopLimit := 255,
                SourceAddress := source, // TEMPLATE PARAMETER
                DestinationAddress := dest // TEMPLATE PARAMETER
 }
Data :=  { UDPHeader := {
                        SourcePort := 777, // NEVERMIND
                        DestinationPort := 521, // SERVICE PORT
                        Length := 0, // CALCULATED BEFORE SENDING
                        Checksum := 0, // CALCULATED BEFORE SENDING
                        Payload := { Command := 1, // RIPng Request
                                     Version := 1,
                                     MustBeZero := 0,
                                     RTE := { // First RTE
                                              IPv6Prefix := P1,
                                              RouteTag := 0,
                                              PrefixLen := PF1,
                                              Metric := 0
                                            },{ // Second RTE
                                              IPv6Prefix := P2,
                                              RouteTag := 0,
                                              PrefixLen := PF2,
                                              Metric := 0
                                            }
                                   }
                      }
      }
}
```

Fig. 9. TTCN-3 template for a RIPng packet

We modeled the protocol version field as a `bistring` field of length four. We expected that integer values (like 6 for the protocol number) would be simply assigned, but they have to be converted to bitstrings. The solution found was to invoke an encoding function that encode the 6 in binary using four digits (`Version := int2bit (6, 4)`). Even though this is not particularly a problem, the solution does not seem natural. It is natural for a developer to expect that the language solves this conversions transparently.

Another relevant limitation found was that we were not able to specify a template with "any number of RTE" (note the difference with a recursive type with any number of RTE). The template shown in figure 9 is defined for a RIPng packet with exactly two RTE. Pattern matching rules embedded in TTCN-3 might allow definition of repetitive parts of structures that might help decreasing the number of data types and templates defined.

Upon message reception, the message classification presented several difficulties, both for handling interleaved reception of RIPng packets and ND ones. This fact conspired against legibility of the test. It is desirable to have some aggre-

gation of "similar" packets. In this way, logical separation of message reception and handling would lead to more structured ATS.

Also some kind of inspection of unknown packets shall be provided. Reception message queues are processed sequentially. Upon arrival of a non-matching packet, the reception queue stalls. A "wild-card" default packet matching rule was introduced, but TTCN-3 does not provide methods for inspecting the unknown packet. Reception of unmatched packets was logged and the analysis had to be done with external tools like `ethereal (http://www.ethereal.com/)`, something that was important during test debugging and log analysis.

3.4 Test execution

From the methodological point of view we intended to perform stepwise refinements of our ATS until we produce the definitive one. Spiral patterns or incremental iterations could not be performed the way that they should. The amount of modules and things to be generated delayed the first ETS test production. The delay introduced until we had the first executable version of the test made that several different pieces of testing code had to be debugged at once and also, feedback for refining the test suites was delayed.

The lack or building blocks stopped us from concentrating only on RIPng templates and test cases. Representation of network topology, like routing tables, were needed. The lack of IPv6 extensions or libraries also forced us to model from simple things, like IPv6 packets, to complex behavior like ND algorithms. We are aware that this was our first TTCN-3 implementation, but all the facts suggests that the *test development cycle* was too big and only few iterations could be performed. The figure 10 shows the effective RIPng test development cycle and the main tasks needed for closing it. It is worth comparing our initial test development plan (see figure 3) with the actual work done. Our experience suggests that network layer support from the tool is needed to reduce the gap and, consequently, development overhead.

The tests performed in our laboratory were done against both a `GNU/Linux` system running `Zebra/RIPngd` and `FreeBSD` system running `routed6`. From the test development point of view, Danet's tool gave the required support for analyzing and debugging purposes. From the test execution point of view we found that log information was hard to analyze. One possible reason is that our changes at PCO were not propagated by the tool to the log files. Thus, Link Layer information was stripped from the packets and did not reach log files.

TTCN-3 language and the tool provide adequate support for issuing a verdict, but we found it difficult not only to explain it but to extract information that eases debugging. When testing for conformance, it is important to produce feedback that helps the product improve compliance. We found it difficult to analyze execution traces. They were useful for test suite debugging, but not for SUT conformance debugging.

Five test cases were developed in time for their presentation at Plugtests 2004 and the rest of the test cases were ready and run at the TAHI Interoperability event. Generated tests were successfully executed during Plugtests and

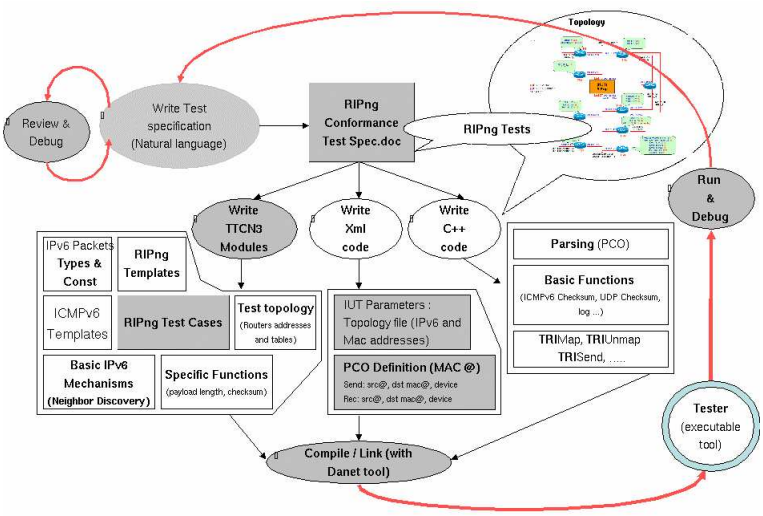

Fig. 10. Test development cycle

the Interoperability event, in October 2004 and January 2005 respectively, with interesting results. But still, we found it not easy to use TTCN-3 tools compared to what we can do with `v6eval`.

4 Main issues

The objective of the present work was to gain experience using TTCN-3 language and tools while addressing an IPv6 test conformance problem. As stated before, one important reason behind the experience was to find provider-independent tools and languages for defining test suites.

We found that there are no standard extensions to handle IPv6 level data. It is also noticeable that there is no explicit support in either TTCN-3 or the tool for lower layer ports, which is required not only for IPv6 testing but for other Ethernet transported protocols. Moreover, there is no easy mechanism for standard IPv6 address representation. For modeling network layer protocols, tester network stack has to be disabled. At that moment all IPv6 implementation details, including packet assembly/disassembly, ND becomes part of our test and had to be emulated. It is desirable that an IPv6 oriented test tool provide as many tools as possible to the expert to help him concentrate on the test purpose. Even though we partially succeeded, our test suites rely on PCO behavior not defined in TTCN-3 standard language. Thus running the tests over an of-the-shelf TTCN-3 tool might be impossible. All our results indicate that it is not possible to provide standard TTCN-3 test suites for IPv6 protocols based on our test architecture built on multiple host emulation from a single test node.

Experimental results suggest that the minor changes performed to the tools would benefit TTCN-3 usage (maybe an IPv6 specialized version of the tools). Field experience supports that the ability of emulating a complex network from a single host is beneficial from the point of view of test execution and is worth considering it as a requirement for the TTCN-3 language.

Ongoing work complementing the RIPng test suite requires usage of other IPv6 features not implemented in the tool. RIPng relies on the IP Authentication Header and IP Encapsulated Security Payload to ensure integrity, authentication and confidentiality of routing exchanges. IPv6 stacks must include IPSec support, used by RIPng, and we have to manually code IPSec from scratch in TTCN-3 for testing SUT security capabilities.

It seems that TTCN-3 template definition was not versatile enough to allow efficient matching of incoming data. It might be interesting to have hierarchical incoming data matching or at least being able to group similar matching rules. This has a direct impact on ATS legibility as the number of entries in matching statements grow considerably. We foresee that the problems of expressiveness would remain also if we use several PTC instead of a single MTC. The impact on ATS legibility is so far unpredictable. The experience of such implementation would help understanding other TTCN-3 aspects, while contrasting single tester *vs* parallel testers on the same matter.

We found no way to define recursive or iterative data templates. Repetitive structures (like routing tables) are sets of individual RTE. The definition of individual templates for packets with one, two, three, etc. RTE again made the code difficult to maintain, unnecessarily large and hard to read. Repetitive tasks, like checksum calculation and verification might be eased if templates would accept dynamic definition (like accepting functions in their definitions).

As a consequence of previous limitations, we were unable to find a pleasant methodology for test creation. It is difficult to abstract parts of the components and protocols for re-using in future implementations. It takes more time than expected to produce runnable ETS. This fact makes that feedback from real execution returns late in test development cycle and the risk of delay due to redesign need is high.

5 Conclusion

We have presented the most important lessons we found when applying the young TTCN-3 language to produce test suites for RIPng protocol. We were able to meet the schedule and the resulting test suite was successfully presented at 50^{th} ETSI Plugtests event and at TAHI Interoperability event. This fact shows that it was possible to develop test suites using TTCN-3, but under special circumstances like having access to the tool source code and changing its implementation.

Success of the language is tightly related to the availability of tools and their capacity to cope in time with the requirements of different fields of application. A careful analysis of enhancement requests has to be combined with pushing

industrial requirements. Widespread availability of tools would speed-up this process.

There are also pending issues regarding language constructs and style that would lead to readable ATS.

Several important decisions were taken without enough study and experimentation. Ongoing work addresses more detailed study of identified problems. In this new stage we are putting special emphasis on all TTCN-3 modular capabilities. Another aspect that is subject of study now is the portability of TTCN-3 code amongst implementations.

Acknowledgments. The authors would like to thank Wolfgang Sachse and Danet technical team for their support during the work. Authors would also like to thank Frederic Roudaut for technical inputs and Bruno Deniaud for giving comments on the paper. Thanks to other members of the TIPI group (`http://www.irisa.fr/tipi`) for all fruitful discussions.

References

1. Andreas Ulrich, Hartmut Köoning, and Thomas Walter. Architectures for testing distributed systems. In Gyula Csopaki, Sarolta Dibuz, and Katalin Tarnay, editors, *(TestCom 1999) Testing of Communicating Systems, Methods and Applications, ISBN 0-7923-8581-0*, pages 93–108. Kluwer Academic Publishers, 1999.
2. Ina Schieferdecker and Theofanis Vassiliou-Gioles. Realizing distributed TTCN-3 test systems with TCI. In Dieter Hogrefe and Anthony Wiles, editors, *(TestCom 2003) Testing of Communicating Systems In 15th IFIP Testing of Communicating Systems, Tools and Techniques, ISBN 3-540-40123-7*, pages 95–109. Springer, 2003.
3. Stephan Schulz and Theofanis Vassiliou-Gioles. Implementation of ttcn-3 test systems using the tri. In Ina Schieferdecker, Hartmut Köning, and Adam Wolisz, editors, *(TestCom 2002) Testing of Communicating Systems, Application to Internet Technologies and Services, ISBN 0-7923-7695-1*, pages 425–442. Kluwer Academic Publishers, 2002.
4. Theofanis Vassiliou-Gioles, Ina Schieferdecker, Marc Born, Mario Winkler, and Mang Li. Configuration and execution support for distributed tests. In Gyula Csopaki, Sarolta Dibuz, and Katalin Tarnay, editors, *(TestCom 1999) Testing of Communicating Systems, Methods and Applications, ISBN 0-7923-8581-0*, pages 61–67. Kluwer Academic Publishers, 1999.
5. Roland Gecse and Sarolta Dibuz. An intuitive ttcn-3 data presentation format. In Dieter Hogrefe and Anthony Wiles, editors, *(TestCom 2003) Testing of Communicating Systems In 15th IFIP Testing of Communicating Systems, Tools and Techniques, ISBN 3-540-40123-7*, pages 63–78. Springer, 2003.
6. Törö. Decision on tester configuration for multiparty testing. In Gyula Csopaki, Sarolta Dibuz, and Katalin Tarnay, editors, *(TestCom 1999) Testing of Communicating Systems, Methods and Applications, ISBN 0-7923-8581-0*, pages 109–128. Kluwer Academic Publishers, 1999.
7. Jens Grabowski and Dieter Hogrefe. Towards the third edition of ttcn. In Gyula Csopaki, Sarolta Dibuz, and Katalin Tarnay, editors, *(TestCom 1999) Testing of Communicating Systems, Methods and Applications, ISBN 0-7923-8581-0*, pages 19–30. Kluwer Academic Publishers, 1999.

8. Jens Grabowski, Anthony Wiles, Colin Willcock, and Dieter Hogrefe. On the design of the new testing language ttcn-3. In Hasan Ural, Robert L. Probert, and Gregor v. Bochmann, editors, *(TestCom 2000) Testing of Communicating Systems, Tools and Techniques, ISBN 0-7923-7921-7*, pages 161–176. Kluwer Academic Publishers, 2000.
9. G. Malkin and R. Minnear. RFC 2080: RIPng for IPv6. http://www.rfc-editor.org/rfc/rfc2080.txt, 1997.
10. ETSI. Es 201 873-1 ttcn-3 core language, version: 2.2.1. http://www.etsi.org/ptcc/TTCN-3%20Downloads/es_20187301v020201p.zip, 2003.
11. Jianping Wu, Whongjie Li, and Xia Yin. Towards Modeling and Testing of IP Routing Protocols. In Dieter Hogrefe and Anthony Wiles, editors, *Testing of Communicating Systems In 15th IFIP Testing of Communicating Systems, ISBN 3-540-40123-7*, pages 49–62. Springer, 2003.
12. T. Narten, E. Nordmark, and W. Simpson. RFC 2461: Neighbor Discovery for IP Version 6 (IPv6). http://www.rfc-editor.org/rfc/rfc2461.txt, 1998.
13. R. Hinden and S. Deering. RFC 2373: IP Version 6 Addressing Architecture. http://www.rfc-editor.org/rfc/rfc2373.txt, 1998.