EXPERIMENTS ON IPV6 TESTING

Tibor Csöndes, Sarolta Dibuz, Péter Krémer

Ericsson Research

Conformance Center, Ericsson Ltd.

H-1037 Budapest, Laborc u. 1., Hungary

E-mail: { Tibor.Csondes, Sarolta.Dibuz, Peter.Kremer } @eth.ericsson.se

Abstract IPv6 lays the groundwork for the next generation of networking, and IPv6 implementations will play vital importance in network-dependent business in the future. This vital role enhances the need for quality and interoperability assurance of IPv6 implementations for which automated testing with test tool – and vendor independent test suites is a promising solution.

The paper presents experiences in testing IPv6 implementations automatically with test cases written in TTCN. The paper also shows via examples how the conformance type of testing can be used to analyze errors occurring during interoperability testing.

Keywords: Internet, IPv6, TTCN, Conformance test, Interoperability test

1. INTRODUCTION

The convergence of the telecom and Internet technologies brings up several questions, one of them is bridging the gap between the different testing approaches of the telecom and datacom world. Reliability and robustness of telecommunication products are assured through test campaigns during which the products are conformance tested according to a strictly standardized process and in most of the cases with standardized test suites.

A test notation was also standardized for writing test tool independent test suites that can be applied by all vendors. Comparable and repeatable test results are gained as the same test suites are used for different implementations and the conditions of testing are described in documents. TTCN (Tree and Tabular Combined Notation) [10], [1] was developed for conformance testing and Abstract Test Suites (ATS) have been standardized for several telecommunication

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35516-0_20](https://doi.org/10.1007/978-0-387-35516-0_20)

protocols in TTCN mainly by ETSI and ITU-T. Most of the telecom vendor companies use TTCN in their internal test procedures because of its usefulness in protocol testing. Tools exist supporting the writing and execution of TTCN test cases as well as generating them from formal specifications written in SDL (Specification Description Language) or MSC (Message Sequence Chart).

Telecom service providers are also executing conformance testing on the products they buy to check if it fulfills the requirements described in the protocol standards. This is a main step assuring the interoperability of the heterogeneous products built in their network. In the datacom world this kind of strict testing is not applied though interoperability of products of different vendors is becoming to be as important as for telecom networks with the growing number of datacom product vendors and the growing reliability requirements. The interoperability events use ad hoc test scenarios, and no test results are really available for the potential customers to support their decision which product to buy.

This paper presents an application of the conformance testing methodology on IPv6 testing. In section two we present the main features of IPv6. Section 3 summarizes ideas why conformance testing would be useful for the Internet protocols. In section 4 we describe the test configuration we have used, in section 5 the test cases and in section 6 the test results of conformance testing of some IPv6 implementations. In section 7 we compare the test results with interoperability problems of the examined IPv6 implementations. Section 8 is the conclusion.

2. **IPv6 OVERVIEW**

IP version 6 (IPv6) [3] is a new version of the Internet Protocol designed as the successor to IP version 4 (IPv4). The changes from IPv4 to IPv6 fall primarily into the following categories:

- *Expanded Addressing Capabilities:* IPv6 increases the IP address size from 32 bits to 128 bits (increase the address space by a factor of 2^{96}), to support more levels of addressing hierarchy, a much greater number of addressable nodes, and simpler auto-configuration of addresses. The scalability of multicast routing is improved by adding a "scope" field to multicast addresses. A new type of address called an "anycast address" is defined, used to send a packet to any one of a group of nodes.
- *Header Format Simplification:* Some IPv4 header fields have been made optional or have been dropped, to reduce the common-case processing cost of packet handling and to limit the bandwidth cost of the IPv6 header. This simplifies and speeds up router processing of IPv6 packets compared to IPv4 datagrams. The IPv6 packet header is fixed-length whereas the IPv4 header is variable-length. Again, the IPv6 design simplifies processing.

- *Improved Support for Extensions and Options:* Changes in the way IP header options are encoded allows for more efficient forwarding, less stringent limits on the length of options, and greater flexibility for introducing new options in the future.
- *Flow Labeling Capability:* A new capability is added to enable the labeling of packets belonging to particular traffic "flows" for which the sender requests special handling, such as non-default quality of service or "real-time" service. IPv4 provides minimal assistance in this area.
- *Authentication and Privacy Capabilities:* Extensions to support authentication, data integrity, and (optional) data confidentiality are specified for IPv6. IPv4 provides no security capabilities other than an optional security label field.

2.1 IPv6 Header Format

An IPv6 protocol data unit has the general form shown in Fig. 1.

Figure 1. IPv6 PDU form

The only header that is required is referred to simply as the IPv6 header (Fig. 2). This is of fixed size with a length of 40 octets, compared to 20 octets for the mandatory portion of the IPv4 header.

Ver	Class	Flow Control	
Payload Length		Next Head	Hop Limit
Source Address			
Destination Address			

Figure 2. IPv6 Header

The fixed 40 octets length of the IPv6 header consists of the following fields (Fig. 2):

- *Version (4 bits)*: IP version number; the value is 6.
- *Traffic class (8 bits)*: The Class field lets the originating host or the forwarding router identify the class or priority of the packet.
- *Flow label (20 bits)*: The Flow Control field lets the source host label a sequence of packets (i.e., a flow) that requires special handling by intermediate routers when the packets travel from source to destination.
- *Payload length (16 bits)*: Length of the remainder of the IPv6 packet following the header, in octets. In other words, this is the total length of all the extension headers plus the transport-level PDU.
- *Next header (8 bits)*: Identifies the type of header immediately following the IPv6 header.
- *Hop limit (8 bits)*: The remaining number of allowable hops for this packet. The hop limit is set to some desired maximum value by the source, and decremented by 1 by each node that forwards the packet. The packet is discarded if hop limit is decremented to zero. This is a simplification over the processing implied for the time-to-live field of IPv4. The consensus was that the extra effort in accounting for time intervals in IPv4 added no significant value to the protocol. In fact, IPv4 routers, as a general rule, treat the time-to-live field as a hop limit field.
- *Source address (128 bits)*: The address of the originator of the packet.
- *Destination address (128 bits)*: The address of the intended recipient of the packet. This may not in fact be the intended ultimate destination if a *Routing header* is present.

2.2 **IPv6 Extension Headers**

The following extension headers have been defined (the numbers show the suggested order):

1. *Hop-by-hop options header*: Defines special options that require hop-by-hop processing
2. *Destination options header*¹: Contains optional information to be examined by the destination node
3. *Routing header*: Provides extended routing, similar to IPv4 source routing
4. *Fragment header*: Contains fragmentation and reassembly information
5. *Authentication header*: Provides packet integrity and authentication
6. *Encapsulating security payload header*: Provides privacy
7. *Destination options header*²: For options to be processed only by the final destination of the packet

In Fig. 3 we can find an example of an IPv6 packet's extension headers.

Figure 3. IPv6 packet's extension headers

2.3 Internet Control Message Protocol Version 6 (ICMPv6)

ICMPv6 [2] is the control protocol of IPv6. It contains error and informational messages. One group of the messages describe the different error types (the numbers indicate the message types):

- 1 Destination Unreachable
- 2 Packet Too Big
- 3 Time Exceeded or Hop Limit Exceeded
- 4 Parameter Problem

The informational messages are mainly used for diagnostic purposes (the numbers indicate the message types):

- 128 Echo Request
- 129 Echo Reply

A **Destination Unreachable** message should be generated by a router, or by the IPv6 layer in the originating node, in response to a packet that cannot be delivered to its destination address for reasons other than congestion.

A **Packet Too Big** must be sent by a router in response to a packet that it cannot forward because the packet is larger than the MTU (Maximum Transmission Unit) of the outgoing link. The information in this message is used as part of the Path MTU Discovery process [8].

If a router receives a packet with a `Hop Limit` of zero, or a router decrements a packet's `Hop Limit` to zero, it must discard the packet and send an ICMPv6 **Time Exceeded** message with `Code` 0 to the source of the packet. This indicates either a routing loop or an initial too small `Hop Limit` value.

If an IPv6 node processing a packet finds a problem with a field in the IPv6 header or extension headers such that it cannot complete processing the packet, it must discard the packet and should send an ICMPv6 **Parameter Problem** message to the packet's source, indicating the type and location of the problem.

Every node must implement an ICMPv6 Echo responder function that receives **Echo Requests** and sends corresponding **Echo Replies**. A node should also implement an application-layer interface for sending **Echo Requests** and receiving **Echo Replies**, for diagnostic purposes.

3. ADVANTAGES OF CONFORMANCE TESTING IPV6

There has been trials for applying conformance testing for Internet protocols [7], [9], [4], [5] and these papers show that it is possible to use the methodology with minor modifications for non-OSI protocols like those Internet protocols described in the above mentioned references. On interoperability events the cause of several errors could be found more easily if conformance test was executed on every product beforehand. This would make interoperability testing more efficient.

This is true for testing IPv6 implementations as well, and as these network elements will play a key role in next generation networks their conformance to the protocol requirements is also a key issue. To achieve a smooth IPv4 to IPv6 migration it is essential that the IPv6 nodes support those features that enable them to communicate with IPv4 ones. This can be assured with executing conformance tests on the appropriate features.

There are initiatives already existing for conformance testing of IPv6. The TAHI project [12] develops conformance and interoperability tests for IPv6. Unfortunately, most of the documentations are in Japanese. Only partial information is available in English. The test configuration consists of one testing node, which has to have multiple connection to IUT (two different Ethernet segments and one serial line). There are some other minor restrictions on IUT, such as its global address, root password, shell and prompt. These limitations make the tests platform specific and the execution more difficult.

Another interesting approach is made by [13], which described test purposes for IPv6 in a similar way as ETSI produces the Test Suite Structure and Test Purpose standards for protocols. These test purposes can be implemented in any language a test tool can execute. Using a standardized test notation like TTCN gives the advantage to be able to execute the test with several test tools.

4. TEST METHODS AND CONFIGURATION

We have used the remote abstract test method that do not contain Upper Tester. We have taken the advantage of open source operating systems like Linux. We have developed a kernel module, which helped us to create and capture any kind of Ethernet packets (i.e. an IPv6 packet with non-existing Ethernet addresses). The main difference to packet filters is that instead of making copies of the original packets, this module redefines a packet's route in the kernel. Doing so we can work with the "original" packets and do not have to cope with packet losses, filter errors or other ambiguities.

For the execution of our tests we have used System Certification System (SCS), which is a TTCN test executor developed in Ericsson. SCS consists of a set of tools: TTCN Translator translates MP files into EXTEL format, which

is then used by Test Component Executor (TCE), which interprets and executes test cases selected from TTCN Manager. The most interesting feature of SCS is the Test Port concept. All Abstract Service Primitives (ASPs) of a given Point of Control and Observation (PCO) have to be mapped onto the Service Primitives (SPs) of the underlying service provider in order to exchange data with IUT. Since different protocols can be based on different services, which use different types of connections. This mapping needs serious programming skills. In order to simplify the task, SCS provides the so called Test Ports. A Test Port is a piece of software that holds routines, which are called after an ASP is received from or before ASPs are sent to TCE. Thus, we get the full control over the SPs.

Figure 4. The structure of SCS

We have developed a new test port for IPv6 tests, which can provide two ASPs for communication with the socket interface: "recv" can only be received, and "send" that can only be sent. IPv6 PDUs are carried by these ASPs through the Test Port. The checksums are calculated in the Test Port, so we do not have to waste time on defining them in the constraints.

5. TEST CASES

The main objective of this work is to show the applicability of TTCN in the field of testing Internet Protocols, especially IPv6. The test purposes for the base specification of IPv6 are written in [13]. It contains 43 test purposes (for the base specification) altogether, which we have arranged into the following five groups:

1. Extension header processing (9 test cases)
2. Option processing (11 test cases)

3. Routing header processing (7 test cases)
4. Reassembly (12 test cases)
5. Fragment header processing (4 test cases)

We have written TTCN test cases for these test purposes. Besides the correct behaviour they contain test cases for invalid events (inopportune test cases are not developed yet). We have developed the necessary tools to be able to execute the tests and evaluated three different IPv6 implementations. Moreover, the next sections will show minor and major problems we have found.

For demonstrating the structure of the test cases let us see an example (Fig. 5). This test case (unrecognized next header) verifies that a node discards a packet containing an unknown value in the `Next Header` field and transmits an ICMPv6 **Parameter Problem** message to the source of the packet.

Test Case Dynamic Behaviour					
Test Case Name : unrecognized_next_header Group : Base_Specification/Host/ Purpose : Configuration : Default : DEFAULT_BASE (TR1_lila_addr, TR1_addr, '1'B) Comments :					
Nr	Label	Behaviour Description	Constraints Ref	Verdict	Comments
1		+RESET			
2		+GEN_PREAMBLE (TR1_lila_addr, TR1_addr, TR1_addr_solnode, '1'B)			
3		IPv6 ! send (paylen:=0) START T1s	s_head (TR1_lila_addr, NUT_lila_addr, ipv6_header_s (138, TR1_addr, NUT_addr))		
4		IPv6 ? recv CANCEL T1s	r_head_data (NUT_lila_addr, TR1_lila_addr, ipv6_header_r (NH_ICMPV6, NUT_addr, TR1_addr), parameter_problem_r (1, 6))	(P)	
5		+GEN_POSTAMBLE (TR1_lila_addr, TR1_addr)			
6		?TIMEOUT T1s		(F)	
7		+GEN_POSTAMBLE (TR1_lila_addr, TR1_addr)			
Detailed Comments :					

Figure 5. Example of a Test Case behaviour

The structure of the test case is very simple, we were following the traditional OSI concept: Preamble, Test Body, Postamble, and - of course - the Default. First, we reset the IPv6 test port and then start the GEN_PREAMBLE test step, which is generally used in every test case. It contains the following:

- sending a **Router Advertisement** message,
- sending an **Echo Request** message (containing 1400 bytes of data),
- upon receiving a **Group Membership Report** or a **Group Membership Reduction** message we discard it,
- upon receiving a **Neighbor Solicitation** message we send the appropriate **Neighbor Advertisement** message (with valid IPv6 and link layer addresses),
- waiting for an **Echo Reply** message, which contains the very same values in the fields of `Identifier`, `Sequence Number` and `Echo Data` as we sent out.

In this preamble we inform IUT that a new router (a TTCN test component) is attached to its segment. To be able to address each other the exchange of the link layer (Ethernet) addresses is necessary, this is done by **Neighbor Solicitation** and **Neighbor Advertisement** messages. The arrival of the **Echo Reply** assures us that we can communicate with IUT and test is ready to begin.

In the main body, after the preamble, we send an IPv6 Header with the unknown value of 138 in the `Next Header` field. Then, we wait for a **Parameter Problem** message whose ICMPv6 `Code` field equals to 1 (unrecognized `Next Header` type encountered) and ICMPv6 `Pointer` field equals to 6 (offset of the `Next Header`).

At the end we close the test case with GEN_POSTAMBLE, which sends a **Router Advertisement** message to the so-called "All-Nodes" multicast address. In this message, a value of zero in the `Router Lifetime` field indicates that this router cannot be used any more as a default router. This only message cleans up the whole mess that was caused by the test case.

The DEFAULT_BASE step is also generally used in almost every test case like GEN_PREAMBLE. In this default step we are waiting for a **Neighbor Solicitation** message and after receiving it we send back an appropriate **Neighbor Advertisement**. This procedure is important in default step because an IPv6 implementation can send **Neighbor Solicitation** messages at any time – in order to ensure its neighbors' reachability – and we have to reply immediately.

6. CONFORMANCE TEST RESULTS

In this section we give detailed description on the results of the test execution on three different platforms (FreeBSD, Linux, Solaris). There is not too much to say about the passed test cases, so we will mainly expound the test cases that failed. The summary of the results is shown in Table 1.

Table 1. Results of Test Cases

Test Groups	Platforms		FreeBSD		Linux		Solaris	
	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL
Extension header processing	9		8	1	8	1		
Option processing	11		11		11			
Routing header processing	6	1	5	2	3	4		
Reassembly	12		6	6	11	1		
Fragment header processing	4		3	1	4			
Total	42	1	33	10	37	6		

6.1 **FreeBSD**

FreeBSD 2.2.8 was used together with `KAME kit RELEASE 19991213`. This was the most reliable and most usable implementation. The installation and configuration was also the easiest. Only one test case gave fail verdict during the execution. The purpose of the test case is to wait for a **Hop Limit Exceeded** message after sending a packet with `Hop Limit = 1` and its *Routing header* contains two more segments. The implementation does not forward the packet (which is the correct behaviour) but omits to send back the appropriate error message. Note that none of the implementations have passed this test case.

6.2 **Linux**

The Linux kernel version 2.2.12 was used together with net-tools-1.52 (it contains the necessary tools for network configuration).

Unfortunately, Linux has major deficiencies in the process of the *Routing header*. Let us see two examples: a node must discard a packet whose *Routing header* contains a multicast address at any position. Contrary to this, Linux forwarded the packet to a multicast address. None of the examined implementations send back a **Hop Limit Exceeded** message when they receive a packet with the value of 1 in the `Hop Limit` field. Linux processed the *Routing header* because it sent back **Parameter Problem** messages with correct values but not always correctly (see the examples above).

There are problems with the reassembly process as well. The implementation recognized the situations where the `Fragment ID` field, the source address or the destination address¹ of the fragments were different.

Linux was also able to handle out-of-order packets but failed one test case according to the inappropriate use of the `m` bit (this flag indicates the existence

¹A node should have more than one address and must differentiate one from the other. If it receives one half of an **Echo Request** to one of its addresses and the other half of the message to another of its addresses, an **Echo Reply** should not be sent.

of more fragments) in the *Fragment header*. A fragmented packet is acceptable if it consists of only one fragment, which means that both the `m` bit and the `Fragment Offset` field has a value of 0. A proper implementation must send an **Echo Reply** to a single-fragmented **Echo Request** but Linux did not behave so.

The most serious problems were generated by the "Time Exceeded" timer. Normally, it is set to 60 seconds, which means that all segments must be received within 1 minute. On the reception of the first segment of a fragmented packet every node starts a 60 second-long timer. When this timer expires and the node did not received all the segments a **Time Exceeded** message must be sent. During the execution we measured 33, 34 and 36 seconds instead of 60 seconds.

This implementation did not keep an eye on the size of the reassembled packet. A `Payload Length` of 278 and a `Fragment Offset` of 65528 would exceed 65535 (which is the maximum size of an IP packet) but Linux did not send a **Parameter Problem** message in this case. The implementation did not send this message and neither did it when a fragment contained an unknown `Destination Option`. However, Linux responded with a **Parameter Problem** message to the same invalid option when it was in a single packet.

6.3 Solaris

Solaris 7.0 (SunOS 5.7) was extended with patch 107788-01, which contains the prototype of IPv6 core functionality. The prototype was stable and worked very well except the handling of *Routing headers* and packet forwarding.

The implementation is not able to recognize invalid or erroneous values in the following fields: `Routing Type`, `Segments Left`, `Header Extension Length`. Solaris passed only those *Routing header* related test cases which did not require the process of the *Routing header* at all. *Routing header* appears in nine test cases and only three of them gave `PASS` verdict (the rest failed).

Two of the passed test cases contain zero in the `Segments Left` field, which means that there are no useful information in the *Routing header* any more (the packet already reached the given destinations). A correct implementation must skip the *Routing header* and proceed to process the next header. The only difference between the two test cases is that in one case the `Routing Type` contains a valid value, in the other case it contains an invalid value. The correct behaviour is the same, an error-free implementation must not send a **Parameter Problem** message in the latter case. There is a multicast address in the *Routing header* in the third test case. The RFC says that a node must silently discard such a packet.

It can be seen very easily that among the above mentioned conditions a correct implementation and a faulty one (which skips every *Routing header*

irrespective of its content) behave the very same. Moreover, this implementation has not forwarded the packets which were correctly source routed, either. Based on these signs we came to the conclusion that the IPv6 prototype of Solaris totally ignores the *Routing header*.

7. **EFFECTS OF THE ERRORS ON INTEROPERABILITY CAPABILITIES**

In this section we show the experienced and the assumed effects of the found errors on interoperability capabilities of each implementation. Since a thorough interoperability test requires much larger network than conformance test, we had to assume the effect of a certain error. In the following we will conclude from the errors found during the conformance testing on possible errors revealed in interoperability testing. We think that some of these bugs would stay undetected using the traditional² interoperability testing.

7.1 **FreeBSD**

The default value of the ``Hop Limit`` field is set by the sender's default routers, which means that the increase of the ``Hop Limit`` by the node itself is not possible. Besides this, an implementation must pass the incoming **Hop Limit Exceeded** messages to the upper-layer process. It can be seen that the degree of the problem depends on the upper-layer process heavily. In case it maintains a timer (or more timers) for the purpose of destination reachability, then the lack of such a message is unable to cause severe problems. But in the other case – when the upper-layer count on this message – the situation could be more dangerous. An interoperability test can detect this fault only in the case of using certain upper-level applications. The analysis of the traces is also possible if the huge amount of traces of *tcpdump* enables.

7.2 **Linux**

There are two problematic fields in Linux's IPv6 code. The first one is the already presented reassembly timer, which is set to 60 seconds in normal case. Imagine the following situation: there is a slow or congested link between the two communicating endpoints and the last fragment of a packet arrives 40 seconds after the first. Linux – as the receiver – sends back a **Time Exceeded** message and drops the arrived fragments after 33-36 seconds. If the delay is permanent this packet will never be delivered! The exploration of this bug

²In this case there are no test scripts and no packet generators. Implementors interconnect their implementations and try them out. The main focus is on interworking, thus they do not examine the correct behaviour either the implementation's answer to faulty attitude or erroneous packets. For example the error qwith the reassembled packet's size in Linux would have been stay undetected.

presumes slow network connection and an application or a method, which is able to split the packets into more fragments.

The most exciting feature of Linux is that it can act as a traffic generator. Sometime Linux floods the network with hundreds of unnecessary packets but only those times when the IPv6 module is loaded into the kernel. We tried other versions of the kernel (2.2.14 and 2.3.47) and re-installed the OS but nothing helped. Very rarely Linux got into an infinite loop, it sent a valid message (for example an **Echo Request**), received the correct answer and started it over again. In other cases it sent out the packet for 3-5 seconds and then stopped. This phenomenon has several impacts: the network is temporarily down and telnet or other types of connections freeze during that time; if the repeated packet is a correct one, then it may totally confuse all computers (including itself as well) on that link; since this symptom occurs rather frequently all connections slow down. Most of the time the user can only see that the network is slow but nothing else. It is quite easy to notice this "feature" since everything slows down and the same packet appears many times in the traces. Certainly, this is the biggest and the most important problem that need to be fixed very soon.

7.3 Solaris

However, Solaris gave more `FAIL` verdict than FreeBSD, all of them were related to the using of the *Routing header* and thus the process of source routing. Note, that source routing is a restriction: a packet must go through given routers or nodes (they are supplied by the *Routing header*). Besides, source routing is not used generally, an implementation can easily communicate with any other node without using it. If the ISP's (Internet Service Provider) policy or the network's topology is not the reason for source routing, the lack of this feature is acceptable. Since this method is rarely used and it is hard to create such packets without a packet generator, the detection of this error in an interoperability test could be very difficult.

8. CONCLUSION

We strongly believe that more thorough testing of datacom products on the protocol requirements defined in the RFCs is becoming more and more important with the growth of the datacom market and with the appearance of IP-based telecommunication networks. Operators used to the traditional telecom market will most probably support the idea of using test suites to trace the capabilities and requirements of products in interoperability events. Using common test suites has also the advantage to produce comparable and repeatable results.

Test suites for the Internet protocols contain usually less test cases so test suites are not as compound as standardized test suites of telecom protocols, like

ISDN. For the success of this test approach in the Internet world the de facto type of test standardization should be worked out to be able to follow the quick changes of the protocol standards with the tests.

TTCN is further developing. TTCNv3 will be standardized in ETSI until the end of 2000. It will be more easy to learn and understand the new version of TTCN and it will be suitable for not only conformance testing but also for other testing types. This shows further possibilities to utilize test suites written for protocol conformance test in function testing and in end-to-end testing of systems. These new features of TTCN will further strengthen its application in testing next generation networks.

References

- [1] B. Baumgarten, A. Giessler: OSI conformance testing methodology and TTCN, North Holland, 1994.
- [2] A. Conta, S. Deering: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, RFC 2463, IETF Network Working Group, December 1998.
- [3] S. Deering, R. Hinden: Internet Protocol Version 6 (IPv6) Specification, RFC 2460, IETF Network Working Group, December 1998.
- [4] R. Gecse: Conformance testing methodology of Internet protocols, Testing of Communicating Systems, Tomsk, Russia, September 1998.
- [5] R. Gecse, P. Krémer: Automated test of TCP congestion control algorithms, Testing of Communicating Systems, Budapest, Hungary, September 1999.
- [6] Robert M. Hinden: IP Next Generation Overview, Communications of the ACM, Vol. 39, No. 6, June 1996.
- [7] T. Kato, T. Ogishi, A. Idoue and K. Suzuki: Design of protocol monitor emulating behaviours of TCP/IP protocols, Testing of Communicating Systems, Cheju Island, Korea, September 1997.
- [8] J. McCann, S. Deering, J. Mogul: Path MTU Discovery for IP version 6, RFC 1981, IETF Network Working Group, August 1996.
- [9] T. Ogishi, A. Idoue, T. Kato and K. Suzuki: Intelligent protocol analyzer for WWW server accesses with exception handling function, Testing of Communicating Systems, Tomsk, Russia, September 1998.
- [10] OSI - Open System Interconnection, Conformance testing methodology and framework, ISO/IEC 9646, 1997.
- [11] William Stalling: IPv6: the New Internet Protocol, <http://www.cs-ipv6.lancs.ac.uk/ipv6/documents/papers/stallings/>
- [12] TAHI project, <http://www.tahi.org/>
- [13] University of New Hampshire, InterOperability Lab: IP Consortium Test Suite, Internet Protocol Version 6, Technical Document, January 2000.