Conformance testing methodology of Internet protocols
Internet application-layer protocol testing – the Hypertext Transfer Protocol

Roland Gecse
Conformance Center, Ericsson Ltd.
H-1037 Budapest, Laborc u. 1., Hungary,
Tel: +36 1 437 7618, Fax: +36 1 437 7219,
e-mail: roland.gecse@lt.eth.ericsson.se

Abstract

This paper examines the applicability of OSI conformance test methodology to Internet protocols. It summarizes the differences between them and introduces the Internet Reference Model along with a new abstract test method, which was designed for the practical purposes of conformance testing of TCP/IP protocols. Some interesting test cases, that were chosen from HTTP, demonstrate the facilities of the model and give impression of testing Internet protocols.

Keywords

conformance test, Internet, HTTP

1 INTRODUCTION

Up to now, in the Internet community, conformance testing was an unknown concept. However, the need for recommendation conforming TCP/IP implementations grows as the application of Internet protocols in business telecommunication systems is becoming reality. It is probable that more and more vendors are going to provide Internet products, whose reliability and interoperability with other products have to be assured.

Although conformance testing methodology (X.290–X.296, 1994-95) was originally intended for OSI based systems, there are ongoing discussions about its applicability to the TCP/IP protocol stack. Numerous articles and conference contributions justify that these questions present a current topic.

(Bil, 1997) founds theoretical base of relay system testing, which is then used, among others, for the testing of Simple Mail Transfer Protocol (Bi2, 1997) and IP router. (Katol, 1997) and (Kato2, 1997) focus on detailed analysis of Transmission Control Protocol's flow control algorithms that are expected to be used on measuring and fixing the majority of implementation problems listed in (Paxton, 1998). On the other hand, (Malek, 1998) deals with interoperability test suite derivation that may be used for the purpose of Internet testing.

The following issues, beside others, will be argued in this paper. Sections 2-5 give an overview of the Internet protocol structure, introduce the Internet Reference Model and suggest a new abstract test method. Also, similarities and differences in layering, data flow and configuration are fetched in comparison to the OSI Basic Reference Model (BRM) (X.200, 1994). After the presentation of a possible test realization (section 6) and a short overview of the Hypertext Transfer Protocol (section 7), section 8 gives some practical testing examples from the field of client, server and proxy testing.

2 COMPARING INTERNET AND OSI ARCHITECTURE

The OSI BRM has 7 layers, each of which with a well-defined task. OSI protocol stacks are designed to fit to this model. The protocol entities (PEs) of a particular protocol suite are associated to the appropriate layers. Peer-to-peer communication between two PEs of the same layer takes place in abstract protocol data units (PDUs) while physical communication with upper and lower layers' PEs is only possible via service primitives (SPs).

Unfortunately, Internet was not planned to have such a detailed abstract model. The structure of TCP/IP, which represents the actual state of Internet has evolved gradually from the beginnings (Carpenter, 1996). Internet has only four layers: link, network, transport and application. Although the general functions of these layers are not as well-defined as OSI's, they provide almost the same functionality. Disregarding that reliable service appears first only in the transport layer, network and transport layers map to their OSI counterparts. Internet link layer maps, in general, to OSI physical and data-link layers. Since the application layer holds all remaining functionality (OSI layers 5-7), applications may gain enormous complexity. Internet protocols do not have standardized SPs, thus in contrast to open systems, the communication between neighboring layers is implementation specific. This, besides the loosely specified layer characteristics, results that layer boundaries are flexible. Another feature that must be kept in mind when talking about Internet is the whole TCP/IP protocol stack should be considered as a single unit together with a set of alternative protocols. The transport layer, for example consists of two protocols: the transmission control protocol (TCP), which is a connection-mode service and the user datagram protocol (UDP) that provides a connectionless service. In a particular communication process, at most one of these services is used.

From the configuration point of view a real open system can act as end system, relay system or both simultaneously. Internet systems have also this kind of configurations with noting that relay systems are called also intermediaries. Intermediaries are further subdivided according to working aspects to proxy, gateway and tunnel systems which will be discussed later.

3 CONFORMANCE TESTING OF INTERNET PROTOCOLS

From the conformance testing perspective it is worth to distinguish between hardware and software implementations. Hardware implementations (eg. IP router, Web-TV equipments) neither implement the whole TCP/IP protocol stack, nor provide interface to protocol layers. Accordingly, they could be examined only by an external test system. Software implementations (eg. FTP client, httpd programs) on the other hand have numerous advantages over hardware systems. Besides the existing test methods (X.290–X.296, 1994-95; Bil, 1997), they imply the possibility of designing more effective new test methods. For the understanding of this methods, a particular TCP/IP implementation should be examined.

4.4BSD-Lite's Net/3 networking code (Wright, 1995) can be considered as a reference implementation of the Internet protocol suite*.

The structure of the Net/3 networking code is presented in figure 1. Application level protocols (FTP, Telnet, RIP) are distinguished from the underlying TCP/IP stack. They are running as processes in the device's user space while underlying layers protocols used to be implemented as a single unit in the operating system space.

The internal structure of this unit consists of three layers: application programming interface (API) or socket layer, protocol layer and interface layer. The public functions of this unit can be reached at the kernel entry points using system calls (SCs) which represent the operating systems' service primitives.

API, in addition to separating the application layer, provides a protocol independent interface to the entities of the underlying protocol layer. It offers a set of different networking features of the kernel that can be reached uniformly via SCs.

The protocol layer holds the Internet transport (UDP, TCP) and network (IP, ICMP, IGMP) layer protocols (Stevens, 1994). The protocol layer does not provide SCs to application layer entities.

The interface layer consists of various device drivers implementing link layer

*Besides TCP/IP, it also supports Xerox Network Systems (XNS), OSI communication protocol families and the Unix domain protocols that are provided for interprocess communication (IPC).

protocols (eg. Ethernet) and procedures that are used for address conversion between the protocol layer and itself. The code for different pseudo devices (loopback interface, BSD packet filter (BPF)) can also be found there. Interface layer functions are accessible through SCs. The packet filtering functions are further applicable for control and observation.

Now, having a global picture of the overall structure of TCP/IP, the Internet Reference Model will be introduced.

4 THE INTERNET REFERENCE MODEL

It can be stated that all of today's software TCP/IP implementations are based upon the architecture of Net/3. By considering this a model will be introduced that is suitable for conformance testing and incorporates the listed features of software implementations.

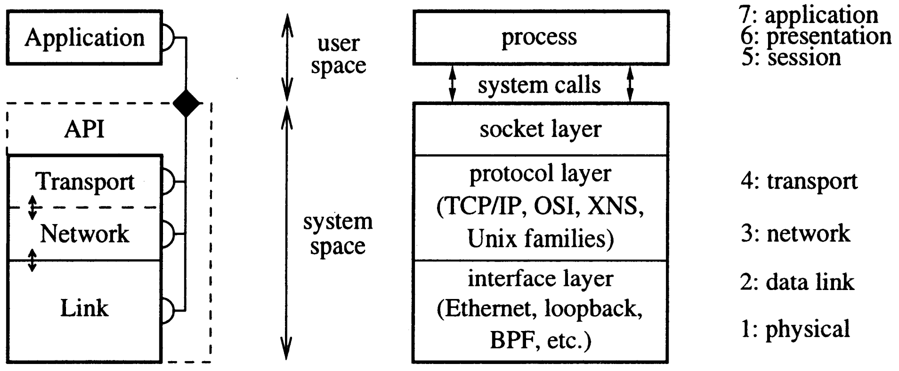

Figure 1 The Internet Reference Model (left), general organization of Net/3 networking code (right)

In the Internet Reference Model (IRM), the functions of SPs are replaced by SCs of API*. These SCs allow applications to send PDUs directly to each layers protocol entity. The API itself should be considered as a switch that connects applications to the selected underlying service via SCs. The functions of the API are provided at kernel entry points (rhombus). The semicircles present the possible destination protocol layers to which SCs provide access. The dashed line expresses that API itself is not a protocol.

Although the IRM has some minor differences from OSI BRM, which are coming from design aspects, the applicability of existing conformance test methodology is straightforward.

*In this context, API is used as a general term, which in a particular implementation (eg. Net/3) stands for both socket API and BPF. That is, because the socket API does not provide access to the interface layer.

5 ABSTRACT TEST METHODS

Considering the open structure of software implementations, the new Joint Test method (JT) will be defined, which can be uniformly applied to testing of all protocols of IRM.

JT can be applied both in Single Party Testing (SPyT) and in Multi Party Testing (MPyT) context. When used in SPyT, it resembles to the local (X.290–X.296, 1994-95) test method. Whereas the MPyT variant has similarities to the local transverse test method in (Bil, 1997).

JT is shown in figure 2, and uses the graphical notation of (Baumgarten, 1994).

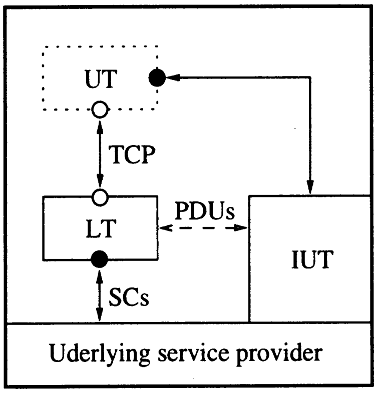

Figure 2 The joint test method.

JT has the following characteristics:

- Test system and system under test (SUT) are on the same system.
- There is an optional Upper Tester (UT), and one Lower Tester (LT) in SPyT; no UT, an arbitrary number (usually 2) of LTs and a Lower Tester Control Function (LTCF) in MPyT. UT, LT(s) and LTCF are application layer processes.
- The Points of Control and Observation (PCOs) are at the LT and UT.
- Test coordination is done using Unix IPC.
- Test events are exchanged in PDUs using SCs of API. The control and observation is provided by means of API.

The most significant difference to the ancestor test methods, which is very advantageous in practical testing of software TCP/IP implementations, is that LT(s), UT and coordination procedures are placed in the application layer regardless of the layer which is occupied by IUT. Another good feature is that JT can be applied to both end systems (SPyT) and relay systems (MPyT), thus intermediaries can be tested out of their environment.

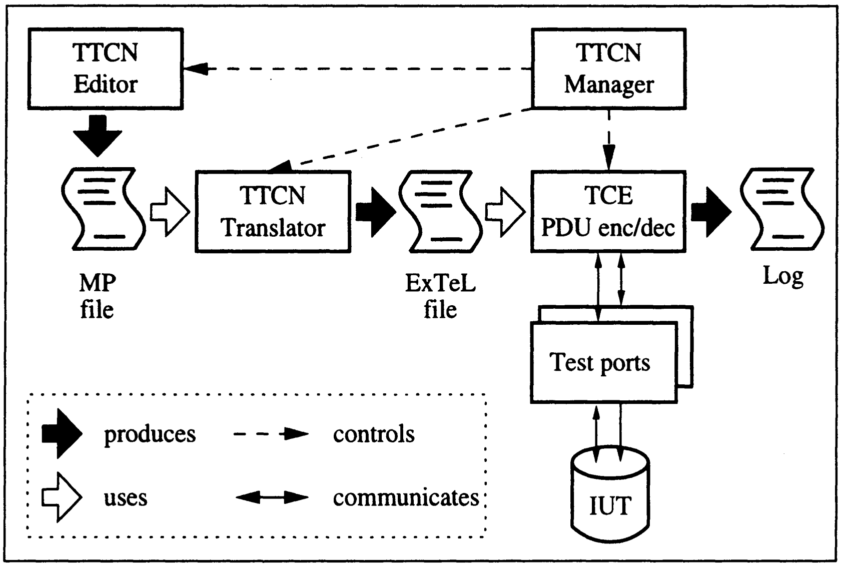

Figure 3 SCS structure.

6 TEST REALIZATION

Having an implementation to be tested and an abstract test suite (ATS), the means of testing should be provided. It consists of the implementation of tester functionalities, the derivation of ATS into executable test suite (ETS) and the production of test documents.

System Certification System (SCS) is a set of tools provided by Ericsson that can be used in a wide variety of testing: functional testing (white-box technique), conformance and interoperability testing (black-box) and performance testing (white/black-box). SCS is based on the following principles:

- Protocol independence. This means that different protocols can be tested on the same manner.
- Multiple simultaneous protocols. Not only one but many protocols can be accessed from the same test.
- Distribution. One test may be distributed (over the Internet), making it possible for each part of the test most closely related to one interface to reside in the box containing that physical interface.
- Platform independence. SCS is independent of the platform in which the SUT executes in. It can execute the same tests both against the physically real SUT and the SUT only simulated in a workstation (bypassing the lowest protocol layers).

One of the main ideas in SCS is that it is an interpreting execution platform.

This means that a TTCN test suite (an MP file) given as input to the Translator is first converted into an intermediate language, ExTeL (Executable Test Language), which then can be directly executed (interpreted) by the ExTeL Test Component Executor, TCE (see also figure 3 above). With this method there is only one phase from a TTCN test suite to the final executable format which makes it different compared to the compiling methods, where an extra compilation and linking phase has to be performed.

Another important feature in SCS is the Test Port concept. With this solution it is possible to develop the core functionality separately without affecting the existing test ports and vice versa.

There exist also two built-in PDU encoder/decoders: BER (Basic Encoding Rules) and a raw binary encoder/decoder.

TTCN Manager is the front end in SCS. It has the control over execution and monitoring. The log files for different test components can be observed in real time.

7 THE HYPERTEXT TRANSFER PROTOCOL

The Hypertext Transfer Protocol (HTTP) (Fielding, 1997) is an application-level protocol for distributed hypermedia information retrieval systems. It is used by World-Wide Web global information initiative since 1990.

HTTP is a generic, stateless, client-server protocol that can be used in wide variety of services by extending through extension of its request methods.

The first version of HTTP - HTTP/0.9 was a simple raw data transfer protocol. HTTP/1.0, as defined by RFC 1945 improved the protocol with many features (MIME-like messages and headers etc.). However, it does not provide enough facilities for handling the effects of hierarchical proxies and virtual hosts. The actual version, HTTP/1.1 offers sophisticated methods for content negotiation, cache control etc.

HTTP has three kinds of communicating parties: client or user agent, origin server and intermediary. There are three common forms of intermediary: proxy, gateway and tunnel.

A proxy is special communication party which, unlike the others, has no OSI equivalent. It can act as both client and server. It may service client requests internally or by passing them on, with possible translation, to other proxies or servers. A gateway receives requests as it were the origin server, and forwards them with possible translation. The client may not be aware that it is communicating with a gateway. A tunnel acts as a blind relay between connections, and is not considered a party of the HTTP communication.

Each party of the communication which is not acting as a tunnel may employ an internal cache for handling requests. The effect of a cache is that the request-response chain is shortened.

In the simplest case communication takes place via a single connection between user agent and origin server. However, more than one connection may be required when intermediaries are present in the request-response chain.

A significant difference between HTTP/1.1 and earlier versions is that persistent connections are the default behavior. Persistent connection means that the connection is not closed after the initial request-response pair. In this case the client can issue further requests. The advantages of persistent connections are: less communication overhead (fewer connections must be set up and released), and thus increased speed. The drawbacks contain: longer duration of connections (origin servers wait for clients to send further requests) and, possibly, conflicts of asynchronous close events (either party of the communication may choose to close the connection any time).

8 TESTING HTTP

8.1 Test documents

The main point of HTTP is information retrieval. World-Wide Web information consists of resources that have to be placed on origin servers and are addressed with Uniform Resource Identifiers (URIs). In order to test HTTP communicating parties, these resources should be provided to the IUT. The resources supplied to them are the following:

- An origin server should have a test Data Base (DB) containing resources that are available for presentation and a set of Configuration files (C) determining its internal operation *.
- A proxy is very similar to the server. It has no local DB, but it usually has and internal cache.
- A user agent has only C, but it also may contain an internal cache.

In addition to putting IUT into a test context (DB and C), the test components also should be familiar with that context. Because of the fact that DB and C play some role of UT, they will be considered so. Thus, their contents must be set up before the test campaign is launched (from test purposes, Protocol Implementation Conformance Statement (PICS), Protocol Implementation Extra Information for Testing (PIXIT)).

If OSI conformance testing methodology should be applied to any protocol of the TCP/IP stack, the Request For Comments should be accompanied by these test documents.

Selected PICS questions and test purposes will be demonstrated below.

*eg.: how many instances of the server should be created; where is the DB located inside the file system; possible mappings that should be applied to various URLs; which documents have to be considered secure and require authentication

8.2 Test configurations

As server, proxy and client have different functions, different test configurations should be defined for testing each of them. Since all common kinds of implementations (except Web-TV-like equipments) are software programs, the best choice is the application of JT. Let's examine the demonstrated configurations deeper!

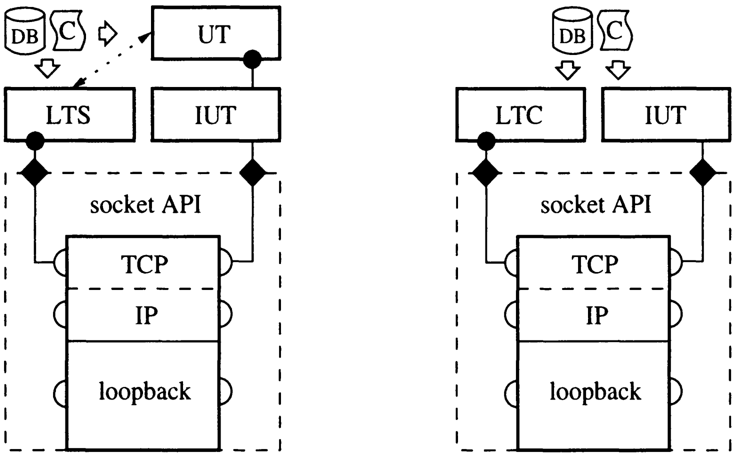

Figure 4 Test configuration for client (left) and server (right) testing.

Figure 4 shows the configuration for testing a user agent. This arrangement consists of an UT and a LT. Their PCOs are denoted with filled circles. The role of UT is played by a user who makes the IUT to issue requests for a certain document. The LT acts as the origin server, it examines and answers the received requests from its DB, according to its C. Test results are determined by the UT and LT together. LT examines whether the client has retrieved the right resource while UT investigates if IUT has presented the resource as well. Test coordination is also done internally.

The configuration for origin server tests can be seen on figure 4. The difference from the client's test configuration is that UT is absent. It has only one PCO at LT. The communication is initiated and the results are examined by LT. Physically, DB belongs to the IUT, however LT also makes use of it.

The configuration for proxy testing (figure 5) is based on the MPyT variant of JT. It has two LTs; LTC plays the client role, while LTS simulates the server.

The test realization happens using SCS. The LTs are implemented as test ports of SCS. In MPyT case, concurrent TTCN is applied. Test coordination between the test components is made using Unix IPC.

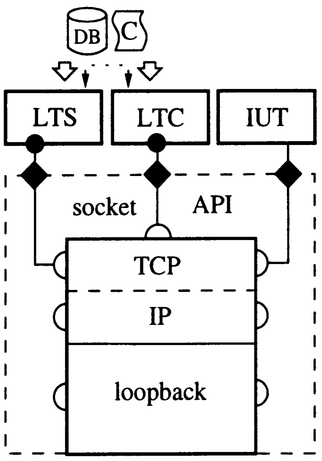

Figure 5 Test configurations for proxy testing.

8.3 Testing client behavior

The behavior of a client or user agent (UA) is, in general, controlled by human using intuitive on-line graphical interface. However HTTP requests that are necessary to retrieve desired resources are generated by the user agent independently of its user.

Let's take a look at a common example: a user browsing the Web, selects an anchor of an HTML page. Doing that, he/she makes the UA issue an HTTP request to the target URL's origin server acquiring an HTML document. After successful downloading, the client parses the document and finds numerous references to inline images and a hyperlink to a style-sheet resource containing essential definitions affecting the layout of the document. The UA should retrieve the required data without any kind of user interaction. Nowadays, HTML pages show many pictures so clients often issue a couple of requests for getting all the contents of a page.

According to the (Fielding, 1997) a client may pipeline its requests. This means it may issue multiple requests without waiting for each of the server's responses. The client, furthermore, should be prepared for expected failure of its attempt eg. when it communicates with an HTTP/1.0 server that neither supports persistent connections nor pipelining, it should cope with such variations. According to this clause, a suggested test purpose follows along with its PICS selection reference.

PICS1.1

Is the IUT (HTTP/1.1 conforming client) able to use pipelining? o YES

TP1.1

To check if the IUT (client) will not pipeline immediately after connecting to the origin server if its last pipeline attempt has failed.

s_tcp_socket ? receive	GET (ROOT, HTTP_1_1, server)	
s_tcp_socket ! send	s200_Ok (ROOT_DOC)	
s_tcp_socket ? receive (res0 := request.request_line.uri)	GET (*, HTTP_1_1, server)	
s_tcp_socket ? receive	GET (*, HTTP_1_1, server)	
s_tcp_socket ? receive	GET (*, HTTP_1_1, server)	
s_tcp_socket ! send	s200_Ok (IUT)	
s_tcp_socket ! close		
START T_SERVER		
s_tcp_socket ? receive	GET (*, HTTP_1_1, server)	
RESET T_SERVER		
? TIMEOUT T_SERVER		(P)

Figure 6 Test case for testing client behavior.

The UT makes the IUT request for a given resource (PIXIT) then the IUT issues the request. The LTS receives the request and sends a 200 'OK' response accompanied with the requested document and does not close the connection, since it waits for the IUT to send further requests. After receiving the response, the IUT parses the document and finds three hyperlinks to inline images. Now, the IUT has exactly three additional requests to issue in order to get the page contents. The LTS waits for the IUT to pipeline these three consecutive requests. After that, the LTS sends the response to the first request along with the connection close message and closes the connection. The IUT should keep track of the status of its requests, and according to the specification it should request for the two unreceived resources automatically. However, its pipeline attempt has failed, so it has to get these resources after one another. If that is the case, the IUT passed the test purpose, otherwise it does not conform to the recommendation (Fielding, 1997). Figure 6 shows the TTCN test case corresponding to the this test purpose (unnecessary timer operations, preambles, postambles, default trees and OTHERWISE statements are removed for better readability).

8.4 Testing Origin Server

There are many common features in an origin server that have to be tested for conformance; the test case that was selected for this demonstration deals with access authentication.

PICS2.1:

Does the IUT (server) provide Basic Access Authentication? o YES

TP2.1:

To verify whether the IUT (server) responds to client requests, which affect protected documents, with 401 'Unauthorized' status, accompanied with the 'WWW-Authenticate' header.

LTC issues a request for a protected resource. The IUT receives the request and finds that the retrieval of the requested document needs authentication. LTC expects IUT to respond with status 401 'Unauthorized'. Moreover, this response should contain the www_authenticate header. In this case the verdict is pass, otherwise the IUT has failed. Figure 7 shows the test case for TP2.1.

c_tcp_socket ! connect		
c_tcp_socket ! send	GET (PROTECTED, HTTP_1_1, IUT)	
START T_SERVER		
c_tcp_socket ? receive [		
response.header_set.response_header.www_authenticate=		
c_Basic_Authentication]		
c_tcp_socket ! close	s401_Unauthorized (IUT)	(P)
CANCEL T_SERVER.		
+timeout		

Figure 7 Test case for testing server behavior.

8.5 Testing proxy

A proxy is the most complicated party of HTTP communication. It receives the clients requests and tries to fulfill them from its cache. If the requested resource cannot be found in the cache or the cached copy is not fresh enough, the proxy retrieves it from another proxy of the cache hierarchy or directly from the origin server.

PICS3.1:
Does the IUT (proxy) employ internal cache for handling requests? o YES
TP3.1:
To verify whether the proxy under test is able to store retrieved resources in its cache.

The test case consists of two parallel test components. The HTTP_C_PTC acts as a client, while the HTTP_S_PTC plays the origin server's role. As a pre-test condition, the proxy's cache should be cleared. Then the two parallel test components are instantiated. The client (LTC) issues a request to the proxy for the ROOT resource located on the server. The proxy gets the request and issues an additional request to the server, since the requested resource cannot be found in its storage. The server sends the response along with the target document to the proxy. If the proxy behaves well, it should store the received document and forward it to the client. After getting the response, the client issues the same request to the proxy. The proxy should fulfill this second request from its cache, it should not turn to the server again. If the server gets another proxy request, then the IUT fails. Otherwise, if the timer expired, the verdict is a conditional pass. If the client gets the new copy of the retrieved document, the verdict is a conditional pass, too. The final verdict is calculated in the postamble. Figure 8 shows the concurrent TTCN (ISO/IEC 9646-3, 1998) test case for test purpose 3.1.

9 CONCLUSIONS

In this paper, differences between OSI and Internet systems were summarized. Then the Internet Reference Model was introduced together with the Joint Test method for conformance testing. Afterwards, a practical application of

CREATE (HTTP_C_PTC : client, HTTP_S_PTC : server		
? DONE (HTTP_C_PTC, HTTP_S_PTC)		
client		
c_tcp_socket ! send	GET (ROOT, HTTP_1_1, server)	
c_tcp_socket ? receive	s200_Ok (IUT)	
c_tcp_socket ! send	GET (ROOT, HTTP_1_1, server)	
c_tcp_socket ? receive	s200_Ok (IUT)	(P)
server		
s_tcp_socket ? receive	GET (ROOT, HTTP_1_1, server)	
s_tcp_socket ! send	RESPONSE (ROOT_DOC)	
START T_SERVER		
s_tcp_socket ? receive	GET (*, *, *)	F
? TIMEOUT T_SERVER		(P)

Figure 8 Test case for testing proxy behavior.

the new concept was demonstrated on the testing of HTTP communication parties.

We have shown a framework for testing Internet protocols, which was worked out on the basis of the conformance testing framework of (X.290–X.296, 1994-95). Experiences with testing HTTP showed that some extensions are necessary to TTCN for making it more suitable to describe test cases for testing Internet protocols. This is true especially for testing performance related features of the product.

Future work can be, for example the interoperability testing based on this concept of Internet protocols, and introduction of formal extensions that are more suitable for Internet testing.

10 ACKNOWLEDGMENTS

The author would like to thank Dr. Sarolta Dibuz and Mazen Malek for their helpful comments, and is also grateful to the anonymous reviewers providing valuable comments and suggestions that have improved the quality of this paper.

REFERENCES

Baumgarten, B. and Giessler, A.: OSI conformance testing methodology and TTCN, North. Holland, 1994.

Bi, J. and Wu, J.: Towards abstract test methods for relay system testing. Testing of Communicating Systems, Volume 10 pp 381-397, IFIP, 1997.

Bi, J. and Wu, J.: Application of a TTCN based conformance test environment on the Internet email protocol. Testing of Communicating Systems, Volume 10 pp 324-330, IFIP, 1997.

Carpenter, B. (editor): Architectural Principles of the Internet, RFC 1958 Informational, IETF Network Working Group, 1996.

Fielding, R., Irvine, UC, Gettys, J., Mogul, J., Frystyk, H. and Berners-Lee, T.: Hypertext Transfer Protocol - HTTP/1.1., RFC 2068 Standards Track, IETF Network Working Group, 1997.

ISO/IEC 9646-3, The Tree and Tabular Combined Notation (TTCN), 1998.

ITU-T X.200, Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model, 1994.

ITU-T X.290–X.296, OSI conformance testing methodology and framework for protocol recommendations for ITU-T applications, 1994-1995.

Kato, T., Ogishi, T., Idoue, A. and Suzuki, K.: Design of Protocol Monitor Emulating Behaviors of TCP/IP Protocols. Testing of Communicating Systems, Volume 10 pp 416-431, IFIP, 1997.

Kato, T., Ogishi, T., Idoue, A. and Suzuki, K.: Intelligent Protocol Analyser with TCP Behavior Emulation for Interoperability Testing of TCP/IP Protocols. Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE X/PSTV XVII '97 pp 449-464, IFIP, 1997.

Malek, M. and Dibuz S.: A Pragmatic Method for Interoperability Test Suite Derivation. EUROMICRO'98, Proceedings of the 24RD Euromicro Conference, Stockholm,Sweden, 1998. Aug 24-26.

Paxton, V. (editor), Allman, M., Dawson, S., Heavens, I. and Volz, B.: Known TCP Implementation Problems, ¡draft-ietf-tcpimpl-prob-02.txt¿ Internet Draft, IETF Network Working Group, May 1998.

Stevens, W. R.: TCP/IP Illustrated Volume 1, The Protocols. Addison-Wesley, 1994.

Wright, G. R. and Stevens, W. R.: TCP/IP Illustrated, Volume 2, The Implementation. Addison-Wesley, 1995.

11 BIOGRAPHY

Roland Gecse is a Ph.D. student of the Department of Telecommunications and Telematics at the Technical University of Budapest. He received his M.Sc. in electrical engineering at Technical University of Budapest in 1996. Since 1997, he is a member of the High Speed Networks laboratory, and joined Conformance Center at Ericsson Hungary. His research interests include conformance, performance and interoperability testing of Internet protocols, formal description techniques and software engineering.