2

PerfTTCN, a TTCN language extension for performance testing

I. Schieferdecker, B. Stepien, A. Rennoch
GMD FOKUS
Hardenbergplatz 2, D-10623 Berlin, Germany,
tel.: (+49 30) 254 99 200, fax: (+49 30) 254 99 202,
e-mail: {Schieferdecker, Stepien, Rennoch}@fokus.gmd.de

Abstract

This paper presents a new approach to test the performance of communication network components such as protocols, services, and applications under normal and overload situations. Performance testing identifies performance levels of the network components for ranges of parameter settings and assesses the measured performance. A performance test suite describes precisely the performance characteristics that have to be measured and procedures how to execute the measurements. In addition, the performance test configuration including the configuration of the network component, the configuration of the network, and the network load characteristics is described. PerfTTCN - an extension of TTCN - is a formalism to describe performance tests in an understandable, unambiguous and re-usable way with the benefit to make performance test results comparable. First results on the description and execution of performance tests will be presented.

Keywords

Performance Testing, Test Suite, TTCN, Quality of Service

Testing of Communicating Systems, M. Kim, S. Kang & K. Hong (Eds)
Published by Chapman & Hall

1 MOTIVATION

Non-functional aspects of today's telecommunication services (e.g. multimedia collaboration, teleteaching, etc.) and in particular Quality-of-Service (QoS) aspects became as important as the functional correctness of telecommunication systems. Different approaches for guaranteeing certain QoS levels to the end users were developed. They include approaches for QoS negotiation between the end users and service and network providers, QoS guarantees of transmission services, QoS monitoring and QoS management, for example in self-adapting applications.

This paper considers QoS in the area of testing. Testing is a general method to check whether a network component meets certain requirements. Network components are considered to be communication protocols, telecommunication services, or end user applications. The requirements on network components are often described in a specification. The tested network component is also called implementation under test (IUT). Testing is either oriented at the conformance of an IUT with respect to the specification of the network component, the interoperability between the IUT and other network components, the quality of service of the IUT, or at its robustness.

QoS testing checks the service quality of the IUT against the QoS requirements of the network component. A specific class of QoS is that of performance-oriented QoS. Performance-oriented QoS requirements include requirements on delays (e.g. for response times), throughputs (e.g. for bulk data transfer), and on rates (e.g. for data loss). We concentrate exclusively on performance-oriented QoS, other classes of QoS are not considered. Subsequently, we use the term performance instead of QoS and refer therefore to performance testing.

One of the well-established methods in testing is that of conformance testing. It is used to check that an implementation meets its functional requirements, i.e. that the IUT is functionally correct. Since conformance testing is aimed at checking only the functional behavior of network components, it lacks in concepts of time and performance. Timers are the only means to impose time periods in the test execution. Timers are used to distinguish between network components that are too slow, too fast or do not react at all. In conformance testing, the correctness of the temporal ordering and exchanged protocol data units (PDUs) or of abstract service primitives (ASPs) have been the main target.

Performance testing is an extension to conformance testing to check also QoS requirements. Performance tests make use of performance measurements. Traditionally, performance measurements in a network consist of sending time stamped packets through a network and of recording delays and throughput. Once measurement samples have been collected, a number of statistics are computed and displayed. However, these statistics are sometimes meaningless since the actual conditions in which these measurements have been performed are unknown.

Different strategies can be used to study performance aspects in a communication network. One consists in attempting to analyze real traffic load in a network and to correlate it with the test results. The other method consists of creating artificial traffic load and of correlating it directly to the behavior that was observed during the performance test. The first method enables one to study the performance of network components under real traffic conditions and to confront unexpected behaviors. The second method allows us to execute more precise measurements, since the conditions of an experiment are fully known and controllable and correlations with observed performance are less fuzzy than with real traffic. Both methods are actually

useful and complementary. A testing cycle should involve both methods: new behaviors are explored with real traffic load and their understanding is further refined with the help of the second method by attempting to reproduce them artificially and to test them. The presented approach to performance testing attempts to address both methods.

This paper presents a new approach to describe performance tests for network components and to test their performance under normal and overload situations. Certain performance levels of an IUT can be identified by means of repeating performance tests with varying parameter settings. On the basis of a thorough analysis, the measured performance can be assessed.

A performance test suite describes precisely the performance characteristics that have to be measured and procedures how to execute the measurements. In addition, a performance test has to describe the configuration of the IUT, the configuration of the network, and the characteristics of the artificial load. The exact description of a test experiment is a prerequisite to make test results repeatable and comparable. The description of the performance test configuration is an integral part of a performance test suite.

The objectives of performance testing can be realized with a variety of existing languages and tools. However, there is only one standardized, well known and widely used notation for the description of conformance tests: TTCN - the tabular and tree combined notation (ISO/IEC 1991, 1996 and Knightson, 1993). In addition, a number of TTCN tools are available. We decided to base our work on TTCN due to its wide acceptance. We define an extension of the TTCN language to handle concepts of performance testing. Only a limited number of additional declarations and functionalities are needed for the definition of performance tests. PerfTTCN - an extension of TTCN with notions of time, traffic loads, performance characteristics and measurements - is a formalism to describe performance tests in an understandable, unambiguous and re-usable way with the benefit to make the test results comparable.

The proposal introduces also a new concept of time in TTCN. The current standard of TTCN considers time exclusively in timers, where the execution of a test can be branched out to an alternative path if a given timer expires. New proposals by Walter and Grabowski (1997) introduce means to impose timing deadlines during the test execution by means of local and global timing constraints. In contrast to that, a performance test gathers measurement samples of occurrence times of selected test events and computes various performance characteristics on the basis of several samples. The computed performance characteristics are then used to check performance constraints, which are based on the QoS criteria for the network component.

Although the approach is quite general, one of its primary goals was the study of the performance of ATM network components. Therefore, the approach is in line with the ATM Forum performance testing specification (ATM Forum, 1997) that defines performance metrics and measurement procedures for the performance at the ATM cell level and the frame level (for layers above the ATM layer).

In this paper, we first discuss the objectives, main concepts, and architectures for performance tests, next we present the language features of PerfTTCN to describe the new concepts, and finally we present some results of experiments on an example handling queries to an HTTP server using a modified test generator of some well known TTCN design tool.

2 INTRODUCTION TO PERFORMANCE TESTING

2.1 Objectives of performance testing

The main objective of performance testing is to test the performance of a network component under normal and overload situations. The normal and overload situations are generated by artificial traffic load on the network component. The traffic load follows traffic patterns of a well-defined traffic model. For performance testing, the conformance of an IUT is assumed. However, since overload may degrade the functional behavior of the IUT to be faulty, care has to be taken to recognize erroneous functional behavior in the process of performance testing.

Another goal of performance testing is to identify performance levels of the IUT for ranges of parameter settings. Several performance tests will be executed with different parameter settings. The testing results are then interpolated in order to adjust that range of parameter value, where the IUT shows a certain performance level.

Finally, if performance-oriented QoS requirements for an IUT are given, performance testing should result in an assessment of the measured performance, whether the network component meets the performance-oriented QoS requirements or not.

The main advantage of the presented method is to describe performance tests unambiguously and to make test results comparable.This is in contrast with informal methods where test measurement results are provided only with a vague description of the measurement configuration, so that it is difficult to re-demonstrate and to compare the results precisely. The presented notation PerfTTCN for performance tests has a well-defined syntax. The operational semantics for PerfTTCN is under development. Once given, it will reduce the possibilities of misinterpretations in setting up a performance test, in executing performance measurements, and in evaluating performance characteristics.

2.2 Concepts of performance testing

This section discusses the basic concepts of the performance test approach. The concepts are separated with respect to the test configuration, measurements and analysis, and test behavior.

2.2.1 Test components

A *performance test* consists of several distributed foreground and background test components. They are coordinated by a main tester, which serves as the control component.

A *foreground test component* realizes the communication with the IUT. It influences the IUT directly by sending and receiving PDUs or ASPs to and respectively from the IUT. That form of discrete interaction of the foreground tester with the IUT is conceptually the same interaction of tester and IUT that is used in conformance testing. The discrete interaction brings the IUT into specific states, from which the performance measurements are executed. Once the IUT is in a state that is under consideration for performance testing, the foreground tester uses a form of continuous interaction with the IUT. It sends a continuous stream of data packets to the IUT in order to emulate the foreground load for the IUT. The foreground load is also called foreground traffic.

A *background test component* generates continuous streams of data to cause load for the network or the network component under test. A background tester does not

directly communicate with the IUT. It only implicitly influences the IUT as it brings the IUT into normal or overload situations. The background traffic is described by means of traffic models. Foreground and background tester may use load generator to generate traffic patterns.

Traffic models describe traffic patterns for continuous streams of data packets with varying interarrival times and varying packet length. An often used model for the description of traffic patterns is that of Markov Modulated Poison Processes (MMPP). We selected this model for traffic description due to its generosity and efficiency. For example, audio and video streams of a number of telecommunication applications as well as pure data streams of file transfer or mailing systems have been described as MMPPs (Onvural, 1994). For the generation of MMPPs traffic patterns, efficient random number generator and an efficient finite state machine logic are needed only. Nonetheless, the performance testing approach is open to other kinds of traffic models.

Points of control and observation (PCOs) are the access points for the foreground and background test components to the interface of the IUT. They offer means to exchange PDUs or ASPs with the IUT and to monitor the occurrence of test events (i.e. to collect the time stamps of test events). A specific application of PCOs is their use for monitoring purposes only. Monitoring is needed to observe for example the artificial load of the background test components, the load of real network components that are not controlled by the performance test or to observe the test events of the foreground test component.

Coordination points (CPs) are used to exchange information between the test components and to coordinate their behavior. In general, the main tester has access via a CP to each of the foreground and background test components.

To sum up, a *performance test* uses an ensemble of foreground and background tester with well-defined traffic models. The test components are controlled by the main tester via coordination points. The performance test accesses the IUT via points of control and observation. A performance test suite defines the conditions under which a performance test is executed. Performance characteristics and measurements define what has to be measured and how. Only a complete performance test suite defines a performance test unambiguously, makes performance test experiments reusable and performance test results comparable.

2.2.2 Performance test configurations

In analogy to conformance testing, different *types of performance test configurations* can be identified. They depend on the characteristics of the network component under test. We distinguish between performance testing the implementation (either in hardware, software, or both) of

- an end-user telecommunication application,
- an end-to-end telecommunication service, or
- a communication protocol.

Of course, additional test configurations for other network components can be defined. The test configurations for these three types of performance tests are given in Figure 1, 2, and 3. The notion System Under Test (SUT) comprises all IUT and network components. For simplification, we omit the inclusion of the main tester in the figures.

The three test configurations differ only in the use of foreground tester. The use of background tester that generate artificial load to the network, and the use of monitors, that measure the actual real load in the network are the same in each of

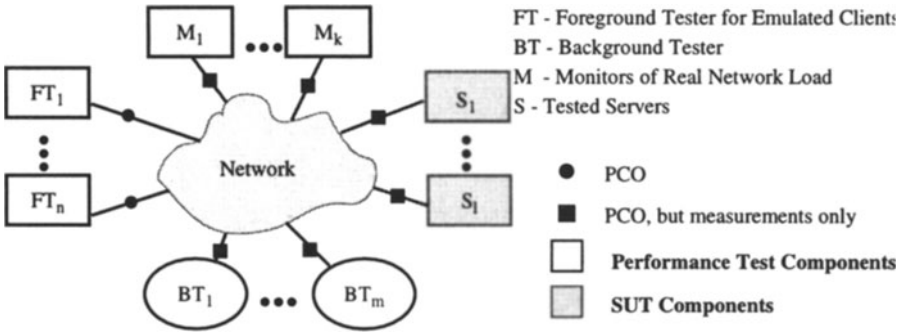

Figure 1 Performance test configuration for a server.

these test configurations.

In the case of performance testing a server in Figure 1, foreground tester emulate the clients. The test configuration for an end-to-end service in Figure 2 includes foreground tester at both ends of the end-to-end service, which emulate the service user. Performance testing of a communication protocol (Figure 3) includes foreground tester at the upper service access point to the protocol under test and at the lower service access point. This test configuration corresponds to the distributed test method in conformance testing (please refer to ISO/IEC, 1991 for other test methods). The service access points are reflected by points of control and observation.

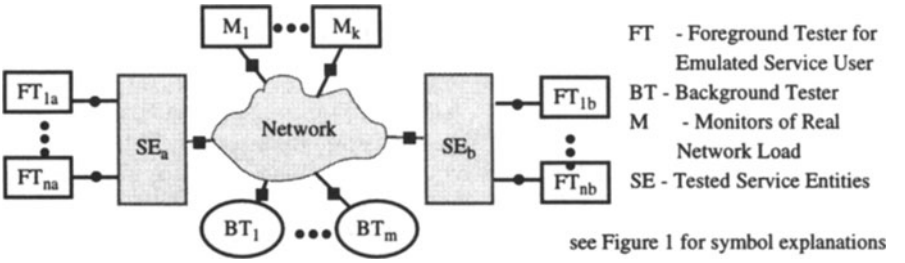

Figure 2 Performance test configuration for an end-to-end service

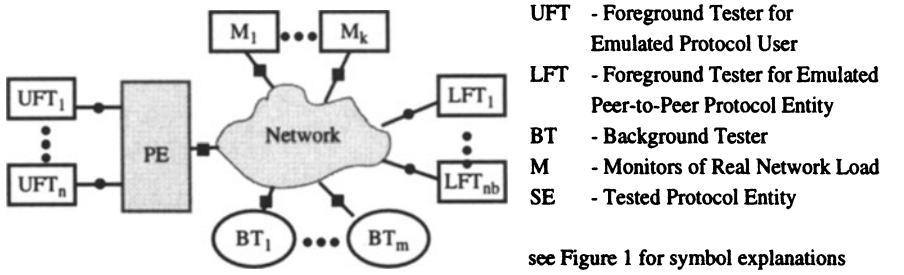

Figure 3 Performance test configuration for a protocol.

2.2.3 Measurements and Analysis

A *measurement* is based on the collection of time stamps of events. A measurement can be executed by monitoring components that are sensitive to specific test events

only. The format of a test event that belongs to a measurement is described by constraints, so that the monitor can collect time stamps whenever an event at a certain PCO matches that format. The constraints used here are the same that are used in conformance testing. A measurement is started once and continues until it is cancelled by a test component or reaches its time duration. Currently, we investigate the need to control measurements explicitly in the dynamic behavior of a performance test.

Based on the measurements, more elaborated *performance characteristics* such as mean, standard deviation, maximum and minimum as well as the distribution functions can be evaluated. Their evaluation is based on predefined *metrics*, which have a well-defined semantics.

Performance characteristics can be evaluated either off-line or on-line. An off-line analysis is executed after the performance test finished and all samples have been collected.

On-line analysis is executed during the performance test and is needed to make use of *performance constraints*. Performance constraints allow us to define requirements on the observed performance characteristics. They can control the execution of a performance test and may even lead to the assignment of final test verdicts and to a premature end of performance tests. For example, if the measured response delay of a server exceeds a critical upper bound, a fail verdict can be assigned immediately and the performance test can finish.

2.2.4 Performance Test Behavior

A performance test suite has to offer features to start and cancel background and foreground test components, to start and cancel measurements, to interact with the IUT and to generate a controlled load to the IUT, as well as to access recent measurements via performance constraints.

At the end of each performance test, a final test verdict such as pass or fail has to be assigned. However, a verdict of a performance test should not only evaluate the observed behavior and performance of the tested network component to be correct or incorrect (i.e. by assigning pass or fail, respectively), but also return the measured performance characteristics that are of importance for the analysis of the test results.

3 PERFTTCN - A PERFORMANCE EXTENSION OF TTCN

This section presents the new language constructs of PerfTTCN for the declaration of traffic models and background traffic, for the declaration of performance measurements, performance characteristics and performance constraints, for the control of test components and measurements, for the use of performance constraints, and for the assignment of verdicts.

3.1 Traffic Models and Background traffic

The location of background test components and the orientation of the background traffic are defined in the test component configuration table (see also Table 1).

For each background test component, PCOs identify the location of the source of the background traffic (left side) and of the destination of the background traffic (right side). Lists of PCOs for the source or destination can be used to declare multipoint-to-multipoint background traffic.

The coordination points of a background test component are used to control its behavior, e.g. to start or to stop the traffic generation. The main tester sends in its dynamic behavior coordination messages to the background test components. The traffic patterns that are generated by a background test component are defined in a

Table 1 Integration of Background Test Components

Test Component Configuration Declaration			
Configuration name: CONFIG_2			
Components Used	**PCOs Used**	**CPs Used**	**Comments**
MTC	PCO_1	CP1,	
PTC1	PCO_2	MCP2, CP1	
Background Test Components			
Identifier	**PCOs Used**	**CPs Used**	**Comments**
traffic1	(PCO_B1) -> (PCO_B2)	BCP1, BCP2	Point to Point
traffic2	(PCO_B1) -> (PCO_B4)	BCP1, BCP2	Point to Point

traffic stream declaration table (see next section).

Specific, implementation dependent details of the location of a background test component, e.g. the connection information such as the VPI/VCI for an ATM connection, are subject to the protocol extra information for testing (PIXIT) document of the performance test suite.

3.2 Traffic models

The purpose of the background traffic is to create load on the communication network that traverses the communication links of the system under test. The background traffic is a continuous, uninterrupted, and predictable stream of packets following a well-defined traffic pattern.

The traffic pattern defines the data packet lengths and interarrival times of the data packets. Traffic patterns can simulate the traffic that is associated with different kinds of applications.

Table 2 MMPP Traffic Model Declaration

Traffic Model Declaration		
Name: on_off **Type**: MMPP **Comments**:		
Length	S1	10
Length	S2	1000
Rate	S1	2
Rate	S2	10
Transition	S1, S2	3
Transition	S2, S1	5

The traffic patterns are defined in traffic model declaration tables (see Table 2 and 3). The declaration selects the stochastic model and sets the corresponding parameters. Each model type has a varying number of parameters and different types of parameters. Therefore, PerfTTCN supports different tables for each type of traffic model. Each traffic model has a name so that it can be referenced later in the traffic stream declaration. Tables 2 and 3 illustrate the table format of an MMPP and CBR model, respectively.

Table 3 CBR Traffic Model Declaration

Traffic Model Declaration	
Name: const1 **Type**: CBR **Comments**:	
PCR	10 MBit/s

The background traffic stream declarations (see also Table 4) relate a traffic stream to a background test component. A traffic stream uses as many instances of a traffic model as necessary to produce significant load. A traffic stream is identified by a name that can be used in the dynamic behavior part to start the corresponding background traffic.

Table 4 Background Traffic Stream Declaration

Background Traffic Stream Declaration			
Traffic Name	**Background Test Component**	**Model Name**	**Nr. of Instances**
Load1	traffic1	on_off	6
Load2	traffic1	const1	2
Load3	traffic2	const1	8

3.3 Measurements and Analysis

The introduction of performance measurements into the testing methodology leads to new tables for the declaration of measurements, performance characteristics and performance constraints as well as to additional operations in the dynamic behavior description of test cases and test steps.

A measurement declaration (Table 5) consists of a metric and is combined with one or two test events that define the critical events of the measurement. For example, for a delay measurement the events define the start and end event. The events are observed only at specific PCOs. The direction of the event is also indicated: "!" means sending and "?" means receiving at that specific PCO (as seen from the test components).

A measurement uses standard metrics such as counter, delay, jitter, frequency, or throughput with predefined semantics. For example, DELAY_FILO is the delay between first bit send and last bit arrived*. User defined metrics (implemented by means of test suite operations) can also be used.

Table 5 Declaration of measurements

Measurement Declaration						
Name	**Metric**	**Unit**	**event 1**	**constr. 1**	**event 2**	**constr. 2**
response_delay	DELAY_FILO	ms	PCO_1 !Request	s_req_spc	PCO_1 ?Response	r_resp_spc

Measurements can be most effectively evaluated with the use of statistical indicators such as means, frequency distributions, maximum, minimums, etc. For that purpose, PerfTTCN offer the concept of performance characteristics. A performance characteristics is declared in a performance characteristics declaration table (see also Table 6). It refers to a single measurement. In order to be statistically significant, a performance characteristics should be calculated only if the measurement has been repeated several times. Therefore, it is possible to define a sample size or a time duration of the measurement for the calculation of a performance characteristic.

Table 6 Declaration of performance characteristics

Performance Characteristics Declaration				
Name	**Calculation**	**Measurement**	**Sample size**	**Duration**
res_delay_mean	MEAN	response_delay	20	
res_delay_max	MAX	response_delay		1 min

*. In general, four different semantics can be given to a delay measurement: FILO = first bit in, last bit out, FIFO = first bit in, first bit out, LIFO = last bit in, first bit out, and LILO = last bit in, last bit out.

3.4 Performance constraints and verdicts

Performance constraints are used for the on-line analysis of observed performance characteristics. For example, if performance falls below some set limits, the verdict should be set to fail. In contrast to constraints in TTCN, a performance constraint evaluation is based on repeated measurement of test events rather than the matching of a single event.

Therefore, we distinguish between functional constraints based on PDU and ASP value matching (that are the traditional constraints in TTCN) and performance constraints. The performance constraint declaration (Table 7) consists of a name and a logical expressions. The expression may use performance characteristics with individual thresholds. More than one performance characteristic can be used in a performance constraint. For example p_resp in Table 7 uses the performance characteristics res_delay_mean and res_delay_max.

Table 7 Declaration of performance constraints

Performance Constraint Declaration		
Name	**Constraint Value Expression**	**Comments**
p_resp	(res_delay_mean < 5) AND (res_delay_max < 10)	
n_p_resp	NOT (p_resp)	

Functional constraints are specified for each event line in the dynamic behavior of a test components. However, performance constraints apply only to the lines where measurements are performed.

3.5 Performance Test Behavior

The behavior of a performance test is specified in the dynamic part of the performance test suite. The main tester is defined in the test cases, while the other test components are specified by test steps.

Test components are created with the START construct. Either they execute their complete behavior or are cancelled explicitly via coordination messages. The control of performance measurements is specified similar to the control of timers, i.e. a measurement can be started and cancelled with START and CANCEL, respectively.

Performance constraints are indicated in the constraint reference column. However, performance constraints are evaluated differently from functional constraints. That is caused by the sample size required for statistical significance and/or the type of metrics used, where more than one observation is required to compute the metric such as the computation of a mean value. Whenever the sample size to evaluate the constraint has not yet been reached, the performance constraint is implicitly evaluated to "true". As soon as the sample size is reached through repeated sampling, the performance constraint is evaluated. If it evaluates to "false", the related event is consequently not accepted. Both, a functional and a performance constraint can be used at the same behavior line. Please note, that performance constraints can be used in qualifiers, too.

Table 8 provides an example of a test case behavior which includes a background test traffic identified by 'Load2', i.e. according to Table 3 it is a constant bit rate. After the background traffic has started (line 1) a series of 'Requests' occurs at PCO_1 (line 2).

The test system awaits from the SUT a 'Response' primitive (line 3 or 5). Due to the response_delay declaration of Table 5 delay measurements occur to determine the time between 'Request' and 'Response'. There are two possibilities to accept

'Response', which are distinguished by the different performance constraints 'p_resp' (line 3) and 'n_p_resp' (line 5). The resulting preliminary test verdict 'pass' or 'inconclusive' depends on these performance constraints.

The test cases finishes when the timer T_response_delay timeouts (line 7). In that case a final verdict is assigned. The reception of an event other than 'Response' terminates the test case (line 8) and measurements, timer, and background traffic are stopped. It is planned to return the measured performance characteristics in combination with the test verdicts in order to support an in-depth result analysis after a performance test finished.

Table 8 The behavior description of a performance test

Test Case Dynamic Behavior					
Test Case Name: www_Get **Group:** **Purpose:** **Configuration:** CONFIG_2 **Default:** **Comments:**					
Nr	**Label**	**Behavior Description**	**Constr. Ref**	**Verdicts**	**Comments**
1		BCP1 ! Start(Load2)			start backgr. traffic Load2
2	top	PCO_1 ! Request START response_delay START T_response_delay	s_req		start measurements
3		PCO_1 ? Response	p_resp	(pass)	acceptable performance
4		GOTO top			
5		PCO_1 ? Response	n_p_resp	(inconc)	unacceptable perf.
6		GOTO top			
7		? T_response_delay CANCEL response_delay			measurement terminates
8		BCP1 ! Stop(Load2)		R	stop background traffic
9		PCO_1 ? OTHERWISE CANCEL response_delay CANCEL T_response_delay		(fail)	unexpected event, stop measurements
10		BCP1 ! Stop(Load2)		R	stop background traffic
Detailed Comments:					

3.6 Comparison with TTCN

Concurrent TTCN has been designed as a test description language for conformance tests, only. It uses discrete test events such as sending and receiving of protocol data units and abstract service primitives. The conformance test suite and the implementation under test interact with each other by sending test events to and receiving test events from the opposite side. A test continues until the tester assigns a test verdict saying that the observed behavior of the implementation conforms (pass) or does not conform (fail) to the specification. In the case that the observed behavior can neither be assessed to be conformant or non-conformant, the inconclusive verdict is assigned. The basis for the development of a conformance test

suite is the functional protocol specification only.

The development of a performance test suite is based on a QoS requirement specification that is combined with the functional specification of the implementation under test. The QoS requirements may include requirements on delays, throughputs, and rates of certain test events. A performance test uses not only discrete test events (those are used to bring the IUT in a controlled way into a well-defined state), but uses also a bulk data transfer from the tester to the IUT. Bulk data transfer is realized by continuous streams of test events and emulates different load situations for the IUT. A performance test assigns not only pass, fail or inconclusive, but also assigns the measured performance characteristics that are the basis for an in-depth analysis of the test results.

The new concepts of PerfTTCN have been introduced in Section 3. The existence of a mapping from PerfTTCN to ConcurrentTTCN would allow us to model performance tests on a level of abstraction that has been specifically defined for performance tests, and would enable us to re-use existing tools for Concurrent TTCN for the execution of performance tests. However, it turned out that some of the new concepts (in particular, traffic models, background tester, measurements, performance constraints) with their semantics can only hardly be represented in Concurrent TTCN. Predefined test suite operations with a given semantics seem to be an easy possibility to include the new concepts. Further study is needed in that area.

4 PERFORMANCE TEST EXAMPLES

Two studies were performed to show the feasibility of PerfTTCN. Performance tests for a SMTP and a HTTP server has been implemented. The experiments were implemented using the Generic Code Interface of the TTCN compiler of ITEX 3.1. (Telelogic, 1996) and a distributed traffic generator VEGA (Kanzow, 1994). VEGA uses MMPPs as traffic models and is a traffic generator software that allows us to generate traffic between a potentially large number of computer pairs using TCP/UDP over IP communication protocols. It is also capable of using ATM adaptation layers for data transmission such as these provided by FORE Systems on the SBA200 ATM adaptors cards. The traffic generated by VEGA follows the traffic pattern of the MMPP models.

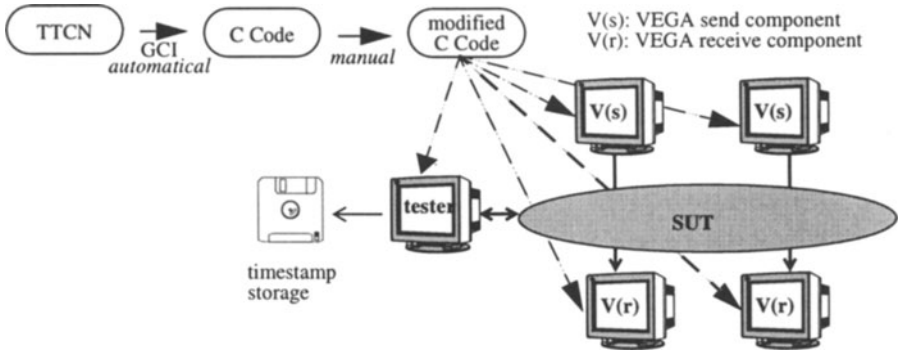

Figure 4 Technical Approach of the experiment.

The C-code for the executable performance tests was first automatically derived from TTCN by ITEX GCI and than manually extended

- to instantiate sender/receiver pairs for background traffic,

- to evaluate inter-arrival times for foreground data packets, and
- to locally measure delays.

Figure 4 illustrates the technical approach of executing performance tests: the derivation of the executable test suite and the performance test configuration. The figure presents also a foreground tester and several send/receive components of VEGA.

The performance tests for SMTP and HTTP server use the concepts of performance test configuration of the network, of the end system, and of the background traffic only. Other concepts such as measurements of real network load, performance constraints and verdicts will be implemented in the next version.

4.1 A performance test for an HTTP server

This example of a performance tests consists of connecting to a Web server using the HTTP protocol and of sending a request to obtain the index.html URL. If the query is correct, a result PDU containing the text of this URL should be received. If the URL is not found either because the queried site does not have a URL of that name or if the name was incorrect, an error PDU reply can be received. Otherwise, unexpected replies can be received. In that case, a fail verdict is assigned.

The SendGet PDU defines an HTTP request. The constraint SGETC defines the *GET /index.html HTTP/1.0* request. A ReceiveResult PDU carries the reply to the request. The constraint RRESULTC of the ReceiveResult PDU matches on "?" to the returned body of the URL: *HTTP/1.0 200 OK*.

The original purely functional test case in TTCN has been extended to perform a measurement of the response time of a Web server to an HTTP Get operation (see also Table 10). The measurement "MeasGet" has been declared to measure the delay between the two events SendGet and ReceiveResult as shown in Table 9.

Table 9 HTTP measurement declaration

Measurement Declaration						
Name	**Metric**	**unit**	**event 1**	**constr. 1**	**event 2**	**constr. 2**
MeasGet	DELAY	ms	SendGet	SGETC	ReceiveResult	RRESULTC

The repeated sampling of the measurement has been implemented using a classical TTCN loop construct to make this operation more visible in this example. The sampling size has been set to 10. The location of the operations due to "MeasGet" measurements are revealed in the comments column in the dynamic behavior of Table 10. It consists in associating a start measurement with the SendGet event and an end measurement with the ReceiveResult event as declared in Table 9. The delay between these two measurements will give us the response time to our request, which includes both network transmission delays and server processing delays.

The main program of the HTTP performance test is shown in Figure 5. The GCI TTCN code of the performance test case is initiated in Line 2. Line 3 instantiates a measurement entity to collect time stamps. The co-working between TTCN GCI and VEGA is initiated by vegaTtcnBridge (Line 4). Models for background traffic are declared and defined on Line 5-7. Background traffic components are declared on Line 8-9. Finally, lines 10-12 define and start the background traffic streams consisting of a background traffic component, a traffic model, and a number of instances. Line 13 starts the performance test case that controls the execution of the test and accesses the measurement entity. The test cases finishes with reporting the measured delays (Line 14). An example of the statistics with and without network load is shown in Figure 6.

This experiment has been performed on an ATM network using Sun workstations and TCP/IP over ATM layers protocols. The graph on the left of Figure 6 shows delay measurement under no traffic load conditions while the graph to the right shows results achieved with six different kinds of CBR and three different kinds of Poisson traffic flows between two pairs of machines communicating over the same segment as the HTTP client machines[†].

Table 10 Performance test case for the HTTP example

Test Case Dynamic Behavior					
Test Case: www_Get **Group:** **Purpose:** **Configuration:** **Default:** **Comments:**					
Nr	**Label**	**Behaviour Description**	**Constr. Ref**	**Verdicts**	**Comments**
1	Top	N ! Connect (NumTimes := 0)	CONNECTC		
2		N ! SendGet START T_Receive START MeasGet	SGETC		start measurement, begin delay sample
3		N ? ReceiveResult (NumTimes := NumTimes+1) CANCEL T_Receive	RRESULTC	(P)	acceptable response, end delay sample
4		[NumTimes < 10] GOTO Top			
5		[NumTimes >= 10] CANCEL MeasGet		R	measurement terminates
6		N ? ReceiveError CANCEL T_Receive CANCEL MeasGet	RERRORC	I	incorrect response
7		? T_Receive CANCEL MeasGet		F	no response
8		N ? OTHERWISE CANCEL T_Receive		F	unexpected response
Detailed Comments:					

5 CONCLUSIONS

The importance of Quality-of-Service aspects in multimedia communication environments and the lack of conformance testing to check performance oriented QoS requirements lead us to the development of a performance testing framework. The paper presents a first approach to extend Concurrent TTCN with performance features.

The main emphasis of our work is the identification and definition of basic concepts for performance testing, the re-usable formulation of performance tests and the development of a performance test run time environment. Thus, the concrete

---

†. Due to lack of space, we have no included the complete performance test suite into the paper. However, it is available on request.

```
1   int main( char* argc, int argv ) { ...
2    GciInit( ); CreatePCOsAndTimers();
3    WWWResponseEnt = new MeasurementEntity("GetWWW"); ...
4    vegaTtcnBridgeInit(argc,argv);
5    backgroungtraffic = new backGroundTraffic();
6    aModel = new vegaModel("cbr_slow", "cbr",10, 0.1, 0);
7    backgroungtraffic->addAModel(aModel); ...
8    aBackGroundDataflow=new BackGroundDataflow("traffic_1",
         "kirk","clyde","udp");
9    backgroungtraffic->addABGDataflow(aBackGroundDataflow); ...
10   aBackGroundTrafficLoad=
         new BackGroundTrafficLoad("traffic_1", "cbr_slow", 3);
11   backgroungtraffic->
         addABGBackGroundTrafficLoad(aBackGroundTrafficLoad);
12   backgroungtraffic->SetupBGTraffic(); ...
13   GciStartTestCase("www_GET"); ...
14   WWWResponseEnt->printStatistics(); ... }
```

Figure 5 Performance test configuration for the HTTP performance test.

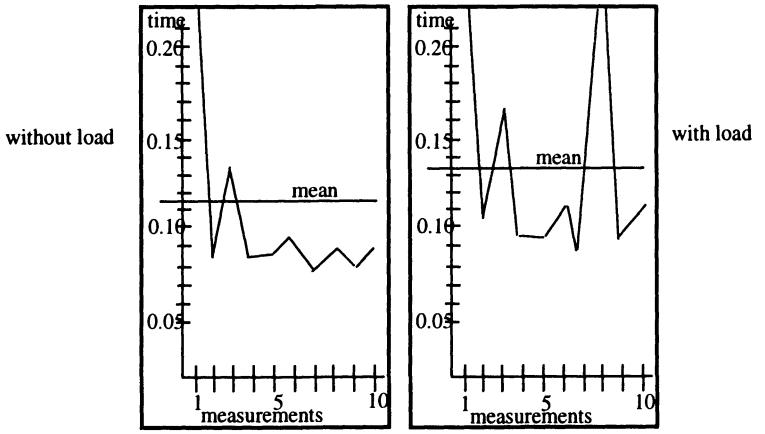

Figure 6 Performance test result of the HTTP example.

syntax of PerfTTCN is a minor concern, but also the basis for ongoing work.

An initial feasibility study of the approach on performance testing has been conducted using the SMTP and the HTTP protocols as examples. The usability of this approach has been demonstrated on a more complex example: A performance test suite to test the end-to-end performance of ATM Adaptation Layer 5 Common Part (AAL5-CP) has been defined only recently (Schieferdecker, Li, Rennoch, 1997).

In parallel, we are further exploring the possibility of re-using existing TTCN tools in a performance test execution environment. Therefore, we are working on a set of test suite operations (reflecting the new performance concepts) and on a mapping from PerfTTCN to TTCN by using these special test suite operations. The definition of the operational semantics of PerfTTCN is currently under work.

6 REFERENCES

Combitech Group Telelogic AB (1996): ITEX 3.1 User Manual.

ISO/IEC (1991): ISO/IEC 9646-1 „Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General Concepts".

ISO/IEC (1996): ISO/IEC 9646-1 „Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The tree and tabular combined notation".

Jain, R. (1996) *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc. Publisher.

Kanzow, P. (1994) Konzepte für Generatoren zur Erzeugung von Verkehrslasten bei ATM-Netzen. MSc-thesis, Technical University Berlin (in german only).

Knightson, K. G. (1993) *OSI Protocol Conformance Testing, IS9646 explained,* McGraw-Hill.

Onvural, R. O. (1994) *Asynchronous Transfer Mode Networks: Performance Issues.* Artech House Inc.

Schieferdecker, I. and Li, M. and Rennoch, A. (1997) An AAL5 Performance Test Suite in PerfTTCN, Proceedings of FBT'97, Berlin, Germany

The ATM Forum Technical Committee (1997): Introduction to ATM Forum Performance Benchmarking Specifications, btd-test-tm-perf.00.01.ps.

Walter, T. and Grabowski, J. (1997) A Proposal for a Real-Time Extension of TTCN, Proceedings of KIVS'97, Braunschweig, Germany.

7 BIOGRAPHY

Ina Schieferdecker studied mathematical computer science at the Humboldt University in Berlin and received her Ph.D. from the Technical University in Berlin in 1994. She attended the postgraduate course on open communication systems at the Technical University in Berlin. Since 1993, she is a researcher at GMD FOKUS - the Research Institute for Open Communication Systems - and a lecturer at Technical University Berlin since 1995. She is working on testing methods for network components, and executes research on formal methods, performance-enhanced specifications and performance analysis.

Bernard Stepien holds a Master degree from the University of Montpellier in France. Subsequently, he carried out research in Transportation Science with the Montreal Transportation Commission and worked as an economist for Bell Canada. He has been a private consultant in computer applications since 1975. He has been active in research on Formal Description techniques with the University of Ottawa since 1985. Currently he is involved in various aspects of communication protocols software with the Canadian Government (Department of Industry, Atomic Energy Control Board), Bell Canada and Nortel.

Axel Rennoch studied mathematics at the Free University of Berlin. His research interests include the application of Formal Description Techniques for testing methodologies and Quality of Service considerations. Currently he is employed as a scientist at the GMD - Research Institute for Open Communication System in Berlin.