PerfTTCN, a TTCN language extension for performance testing

by Ina Schieferdecker, Bernard Stepien[1], Axel Rennoch

GMD-FOKUS, STEP Group

Hardenbergplatz 2, D-10623 Berlin, Germany

http://www.fokus.gmd.de/step

Abstract

This paper presents a new approach to test the performance of protocols, services, or application implementations (IUT) under normal and overload situations, to identify performance levels of the IUT for ranges of parameter settings and to assess the measured performance. A performance test describes precisely the configuration of the IUT, the configuration of the network, and the load characteristics (the so called performance test configuration). PerfTTCN - an extension of TTCN with notions of time, measurements, and performance - is a formalism to describe performance tests in an understandable, unambiguous and re-usable way with the benefit to make the test results comparable.

1. Motivation

Different approaches for guaranteeing the end user certain levels of QoS were developed, since the non-functional aspects, in particular Quality-of-Service (QoS) aspects of distributed tele-communication services (e.g. multimedia collaboration, tele-teaching, etc.) are as important as functional aspects. They include approaches for QoS negotiation between peer user and service and network provider, QoS guarantees of communication services, QoS monitoring and management including self-adapting applications.

This proposal considers QoS in the area of testing. In general, testing is a method to check whether an implementation meets certain requirements that are described in a specification. QoS testing checks the end-to-end quality of a service implementation against the QoS requirements. A specific class of QoS is that of performance-oriented QoS including delays (e.g. of a response), throughputs (e.g. for bulk data), and rates (e.g. of errors). We concentrate exclusively on performance-oriented QoS, other classes of QoS are out of the scope. Subsequently, we use the term performance instead of QoS and consequently performance testing instead of QoS testing.

One of the well-established methods in testing is that of conformance testing, which is used to check that an implementation meets its functional requirements. Since conformance testing is aimed at checking purely the functional behaviour of system implementations, it lacks in concepts of time and performance. Timers are the only way to require certain time periods for test events to occur. They are used to determine whether test events occur too early, too late or not at all.

While traditionally the temporal ordering and type of PDUs/ASPs have been the main target of conformance testing, we attempt here to introduce performance measurements and Quality of Service (QoS) requirements in the conformance testing. Performance measurements in a

1. Guest researcher from the University of Ottawa, Canada

network traditionally consists in sending time stamped packets through a network and record delays and throughput. Once such data has been collected a number of statistics can be computed and displayed. However these statistics can sometime be meaningless when the actual conditions in which these measurements have been performed are unknown. Different strategies can be used to study operational conditions of a network. One consists in attempting to analyse real network traffic load and correlate it with the test results and the other method consists in creating artificially specific network traffic load and correlate it directly to specific behaviours observed in the performance test. The first method enables to study performance under real traffic conditions and confront unexpected behaviours while the second method enables more precise scientific measurements because the conditions of an experiment are fully known and controllable and correlations with observed performance are less fuzzy as with real traffic. Both methods are actually useful and somewhat complementary and a testing cycle involving both methods can be considered by exploring new behaviours with real traffic load and then attempting to reproduce them artificially to further refine their understanding with the help of the second method. The presented approach to performance testing attempts to address both methods.

Although this method is quite general, one of its primary goals is for the study of ATM network performance at the application layer. The approach used in this work is in line with the ATM Forum performance benchmarking specifications [ATM96] that are also interested to measure performance not at the ATM cell level but by frame-level performance and performance perceived at higher layers.

The goals of performance testing can be achieved with a variety of existing languages and tools. TTCN[CMTF, TTCN] is the only standardized, well known and widely used notation for the description of conformance tests, for which a number of 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 performance testing and explore the integration of traffic load models into the TTCN language. In both cases there are limited additional declaration constructs necessary to achieve this goal.

Our proposal introduces a new consideration of time within the TTCN language. So far, the current standard 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 [KIV97] introduce concepts of local and global time constraints during the execution of a dynamic behaviour. Our proposal consists in gathering substantial samples of time measurements between selected test events and compute various statistics on determined sample sizes to be used in the evaluation of performance characteristics. These characteristics are then used to verify their conformance to performance constraints based on QoS criteria. In contrast to current TTCN, a performance constraint evaluation is based on repeated occurrences of pairs of events rather than the evaluation of a single event.

In this paper, we first introduce the objectives, main concepts, and architecture of performance testing, next we present the language features to describe the new concepts in TTCN, 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 primary goal of performance testing is to test the performance of an implementation under normal and overload situations. The normal and overload situations are described with a performance test configuration, which identifies the configuration of the IUT, the configuration of the network, and the load characteristics.

A notation for the description of performance tests is needed, that should have a well-defined syntax and an operational semantics in order to reduce the possibilities of misinterpretations. That notation for performance tests is a precondition to make performance tests understandable, unambiguous and re-usable.

The main advantage of our method is to make the 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. Once more it has to be noted that the mere fact that these tests are described formally ensures unambiguity and re-usability.

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.

For performance testing, the conformance of the IUT is assumed. However, since overload may degrade the functional behaviour of the IUT to be faulty, care has to be taken to capture erroneous functional behaviour in the process of performance testing as it is done in conformance testing.

Finally, if performance requirements for the IUT are given, performance testing should result in an assessment of the measured performance, i.e. does the implementation under test meet the performance requirements or not.

2.2. Concepts of performance testing

This section discusses the basic concepts of the performance test approach. The concepts are separated w.r.t. the test configuration, measurements and analysis, and test behaviour.

2.2.1. Test Components

A *performance test* consists of several, possibly distributed foreground and background test components. They are coordinated by a main tester (control component).

A *foreground test component* realizes the communication with the IUT. It directly influences the IUT. The foreground tester uses discrete test events, which are the same test events that are used in conformance testing, in order to drive the IUT into specific states in which the performance measurements are executed. Once the IUT is in a state that is under consideration for performance testing, the foreground tester issues a continuous stream of data packets to emulate the foreground load for the IUT. The foreground load is also called foreground traffic. It is described by means of traffic models.

A *background test component* generates continuous streams of data in order to emulate the load of the IUT. A background tester does not directly communicate with the IUT. It only implicitly influences the IUT as it brings the IUT into minimal, normal, or overload situations. The background traffic is also described by means of traffic models.

Traffic models describe the communication data as a stochastic stream of data packets with varying interarrival times and varying packet length. An often used model for the description of traffic is that of Markov Modulated Poison Processes (MMPP[ONV94]). We selected this model for traffic description due to its generosity (e.g. a number of different multimedia streams such as audio and video have been described as MMPPs) and efficiency (e.g. 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 SDUs or PDUs with the IUT and to monitor the occurrence of test events (i.e. to collect the time stamps of test events).

A specific use of PCOs are their use for monitoring purposes only. Monitoring only 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.

To sum up, a *performance tests* uses an ensemble of foreground and background tester with the used traffic models and points of control and observation, and points of measurements. The performance test configuration describes unambiguously the conditions under which the measurements are executed. It is the description of the performance test configuration that makes performance test experiments re-usable and performance test measurements comparable.

2.2.2. Performance test configurations

The test components of a performance test may be distributed in the communication network and may use load generators to generate background traffic and monitors to collect time stamps and to execute measurements. A main tester is used to coordinate the test activities of the tester.

One can identify different *classes of performance test configurations* in dependence of the type of the implementation under test. For example in the case of performance testing in communication networks, we distinguish between performance testing the implementation (either in hardware, software, or both) of

- an application,
- an end-to-end transmission service, and
- a protocol.

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 a main tester in the figures.The test configurations differ only in the use of foreground tester. The usage of background tester to generate artificial load to the network, i.e. to bring the network into normal load or overload situations, and measurements on the actual real load in the network, i.e. to monitor the load of real network

applications (as opposed to the artificial background tester), is the same in each of the test configurations.

In the case of performance testing of a server (Fig. 1), foreground tester emulate the clients.

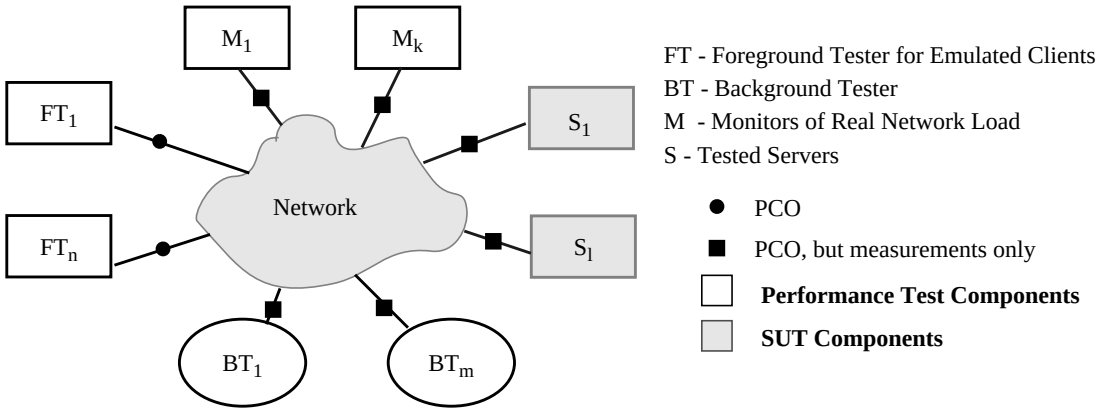


Fig. 1: Performance test configuration for a server

That test configuration for an end-to-end service (Fig. 2) includes foreground tester at both ends of the end-to-end service, which emulate the service user.

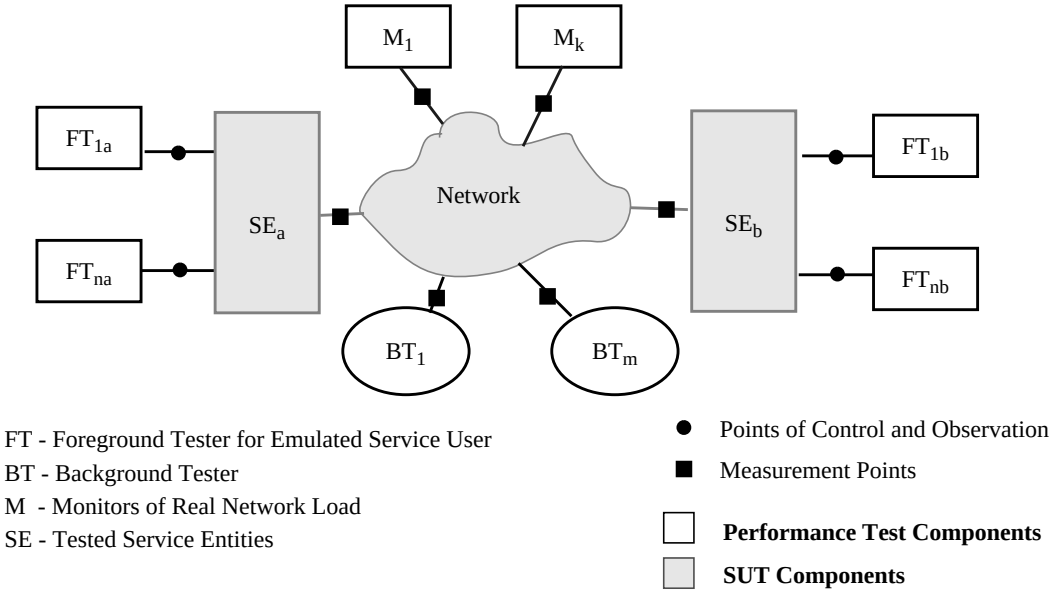


Fig. 2: Performance test configuration for an end-to-end service

Performance testing of a protocol (Fig. 3) with a distributed test method (please refer to [CTMF] for other test methods) includes foreground tester at the upper service access point to the protocol under test and at the lower service access point. The service access points are reflected by points of control and observation.

2.2.3. Measurements and Analysis

A *measurement* is based on the collection of time stamps of events. One has to describe precisely the format of each event that belongs to a measurement, so that the time stamp can be collected whenever an event at a certain PCO matches that format. A measurement is started once and continues until it is explicitly cancelled or reaches the time duration.

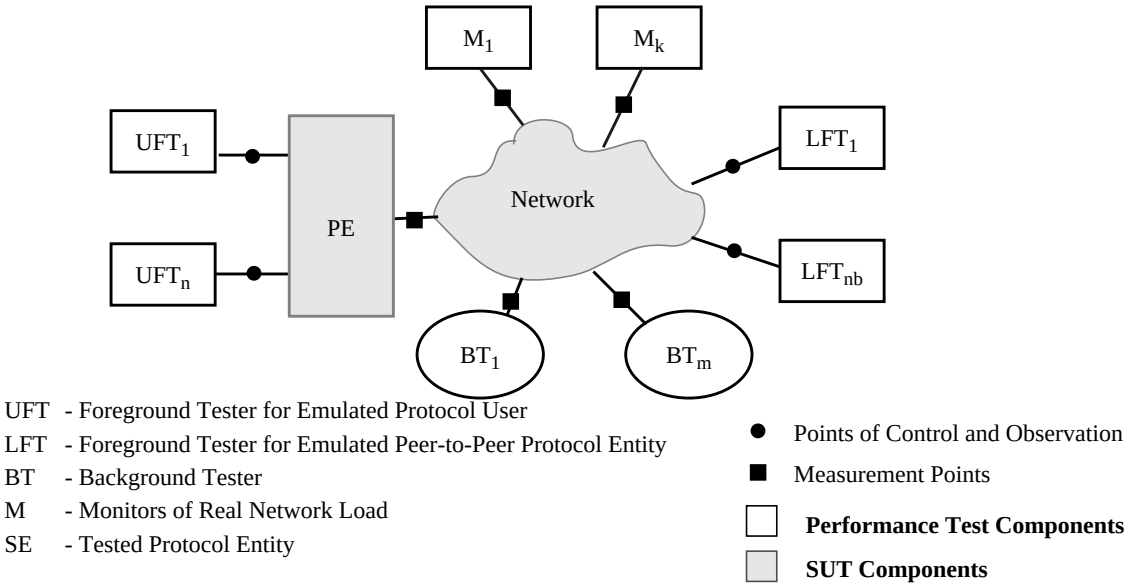


Fig. 3: Performance test configuration for a protocol

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. These are supported by so called *metrics*.

The evaluation of performance characteristics is supported either by an on-line analysis (i.e. during test execution) or in an off-line analysis (i.e. after the test finishes and all samples have been collected). On-line analysis is in particular needed for *performance constraints* that allow one to describe requirements on the observed performance so that either a pass or a fail verdict can be assigned.

2.2.4. Performance Test Behavior

A foreground test components has to offer features on starting and cancelling background test components, on starting and cancelling measurements, on generating controlled load to the IUT, and on accessing the recent measurements via performance constraints. Finally, performance verdicts should be assigned so that they do not only assess the observed behaviour and performance of the IUT to be true or false, but also that give back the measured performance characteristics being of importance for the test report.

3. PerfTTCN - a performance extension of TTCN

The enhancement of conformance testing to performance testing leads also to particular extensions of TTCN. A number of new language constructs have been created to handle

- background traffic declarations
- performance measurement declarations and performance constraints
- control of background test components, measurements, performance constraints and verdicts

3.1. Performance test configurations

Since background test traffic depends on a particular test configuration its localization and orientation is to be added to a test component configuration declaration (Table 1).

For each background test traffic instance PCOs should be defined to identify the location for the test generation (left side) and its destination (right side). In addition, PCO lists are foreseen to integrate multipoint transmissions. The related coordination points are necessary for the control of the traffic generator and any traffic monitor. These components are controlled by coordination messages within the dynamic test behaviour. The qualities of the background traffic are defined by traffic models which are referred in Background Traffic Stream Declarations (see next section). Specific details on e.g. connection information such as VPI/VCI for an ATM connection are subject to the PCO PIXIT document information.

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 Component			
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

Table 1: Integration of Background Test Components

3.2. Traffic models

The purpose of the background traffic is to create load on a network that will be traversed by the communication links of the system under test. The background traffic is a constant uninterrupted and predictable stream of packets following a given stochastic pattern of transmission times.

One of the main characteristics of the background traffic is to follow specific rate patterns that follow stochastic laws. The purpose of these patterns is to simulate the traffic associated with different kinds of applications.

The two essential components of a traffic load are the dataflow configuration and the stochastic models declarations. These two components are then combined in the background traffic stream declarations using as many instances of dataflow/stochastic models combinations as necessary to produce significant load.

This consists in selecting the number of instances of each dataflow/model type combination. Each of these traffic streams is identified by a name that can be used in the dynamic behaviour part to start the appropriate traffic load.

The traffic model declaration is used to select stochastic models and set their corresponding

Background Traffic Stream Declaration			
Traffic Name	**Background Test Component**	**Model Name**	**Number of Instances**
Load1	traffic_1	on_off	6
Load2	traffic_1	const1	2
Load3	traffic_2	const1	8

Table 2: Background Traffic Stream 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

Table 3: MMPP Traffic Model Declaration

parameters. Each model type has a variety of number and kind of parameters. Thus the text version of PerfTTCN has different grammatical constructs for each type of models. Each model selection has also a name so that it can be referenced later in the traffic configuration declarations. The following Tables 3 and 4 illustrate the table format of an MMPP and CBR model, respectively.

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

Table 4: CBR Traffic Model Declaration

3.3. Measurements and Analysis

The introduction of performance measurements comprises new tables for declaration and constraints as well as additional operations within the behaviour description of test cases and test steps.

Any measurement declaration (Table 5) is defined by a metric and combined with two test events (and related constraints) to indicate the start and termination of the measurement. We propose to introduce standard metrics like counter, delay, jitter, frequency, throughput with a predefined semantics (e.g. DELAY with time between first bit send and last bit arrived semantics[2]). Also, user defined metrics (implemented with test suite operations) should be

allowed.

Measurement Declaration							
Name	**Metric**	**Unit**	**event 1**	**constraint 1**	**event 2**	**constraint 2**	**Comments**
response_delay	DELAY	ms	Request	s_req_spc	Response	r_resp_spc	

Table 5: Declaration of measurements

In order to be statistically significant, a measurement needs to be repeated several times. Measurements can be most effectively evaluated with the use of statistical indicators such as means, frequency distributions, maximum, minimums, etc. We propose the declaration of performance characteristics (Table 6) which refer to single measurement declarations. It is possible to define a number of measurements or a time duration for the calculation of a performance characteristic.

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

Table 6: Calculation of performance characteristics

3.4. Performance constraints and verdicts

In PerfTTCN, measurements are performed for the purpose of conformance testing in addition to general performance evaluation. This means that if performance falls below some set limits, the verdict should be set to fail.

Therefore, we distinguish between functional verdicts based on PDU and ASP value matching (that are the traditional constraints in TTCN) and measurement related verdicts. A separate measurement related verdict shall be provided. The performance constraint declaration (Table 7) consists of a name and a logical combination of expressions, where an expression consists of performance characteristics with individual thresholds.More than one performance characterisitic can be used in a performance constraint (e.g. p_resp in Table 7).

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

Table 7: Declaration of performance constraints

2. 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.

While functional constraints are practically specified for each event line of a dynamic behaviour description, performance constraints are not because they apply only to the lines where measurements are performed.

The main difference between functional and a performance constraint is that a performance constraint is not based on a single event but a repeated sample of the same event. This sampling can be achieved by repeated measurement within a single test case. This is when an event can occur several times because it is located within a loop using a goto construct. This also implies that different measurement entities could be found within the scope of a same loop but also that they could belong to different and also overlapping loops with different boundary sizes that would generate different sample sizes.

3.5. Performance Test Behaviour

The control of performance measurements is specified in the behaviour description of test cases or test steps, similar to the control of test timers. Predefined measurements will be executed only if it is stated explicitly in a behaviour description. Furthermore the related “constraint 1” must be valid (see Table 5). Normally the measurement terminates automatically when “event 2” with “constraint 2” occurs. In some (error) situations, measurements could be cancelled too.

Performance constraints are indicated in the constraint reference column. Performance constraints are however evaluated differently from functional constraints because of 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 (mean, standard deviation, etc...). Whenever the sample size to evaluate the constraint has not yet been reached, the performance constraint is implicitly evaluated to “true”. However as soon as, through repeated sampling, this sample size is reached, if the performance constraint is evaluated to “false”, the related event is consequently not accepted. Both a functional (standard) and a performance constraints can be entered for a same line. Performance constraints are meaningful and could be used in qualifiers, too.

Table 8 provides an example of a test case behaviour which includes a background test traffic identified by ‘Load2’, i.e. according to Table 2 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 outcoming 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.

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. Conformance test suite and IUT interact with each other by sending test events to and receiving test events from the opposite side. A test continues until the tester

Test Case Dynamic Behaviour					
Test Case Name: www_Get **Group:** **Purpose:** **Configuration:** CONFIG_2 **Default:** **Comments:**					
Nr	**Label**	**Behaviour Description**	**Constraints Ref**	**Verdicts**	**Comments**
1		BCP1 ! Start(Load2)			start background traffic ‘Load2‘
2	top	PCO_1 ! Request START response_delay START T_response_delay	s_resp		start measurements
3		PCO_1 ? Response	p_resp	(PASS)	acceptable performance
4		GOTO top			
5		PCO_1 ? Response	n_p_resp	(I)	unacceptable performance
6		GOTO top			
7		? T_response_delay CANCEL response_delay			measurement terminates
8		BCP1 ! Stop(Load2)		R[a]	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:					

Table 8: Usage of new performance features in behaviour description

Note a. Please note, that in the next version of PerfTTCN it is planned to return the measured performance characteristics together with the verdicts in order to support an in-depth result analysis.

assigns a test verdict saying that the observed behaviour of the implementation conforms (pass) or does not conform (fail) to the specification. In the case that the observer behaviour 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 may be used to bring the IUT in a controlled manner 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. [GCI] - and a distributed traffic generator with MMPPs as traffic models [VEGA]. VEGA is a traffic generator software that enables to generate traffic between a potentially large number of computer pairs using TCP/ UDP over IP communication protocols. It also has the capability to use ATM native transport protocols such as these provided by FORE for the SBA200 ATM adaptors cards. The traffic generated by VEGA is purely stochastic driven.

The C-code for the executable test suite 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. In figure 4 we illustrate this technical approach: the derivation of the executable test suite and the performance test configuration. We include a foreground tester and several send/receive components of VEGA.

At this point our initial experimentations were performed using performance test configurations of the network, the end system, and of the background traffic only. Other aspects such as measurements for real network load, performance constraints and verdicts will be implemented in the next version.

4.1. A performance test for an HTTP server

This simple example consists in connecting to Web server using the HTTP protocol and sending a request to obtain the index.html URL. If the query is correct we should receive a result PDU containing the text of this URL. If this URL is not found either because the queried site does not have a URL of that name or if the name was incorrect we may receive an error PDU reply, otherwise we might receive unexpected replies and decide that our test has failed.

We have defined a PDU SendGet that encodes our request as: *GET /index.html HTTP/1.0* defined in constraint SGETC and we have defined a ReceiveResult PDU that decodes the reply to our request and matches on the first status line and on “?” for the returned body of the URL: *HTTP/1.0 200 OK* that is followed by the body of the URL response defined in constraint RRESULTC.

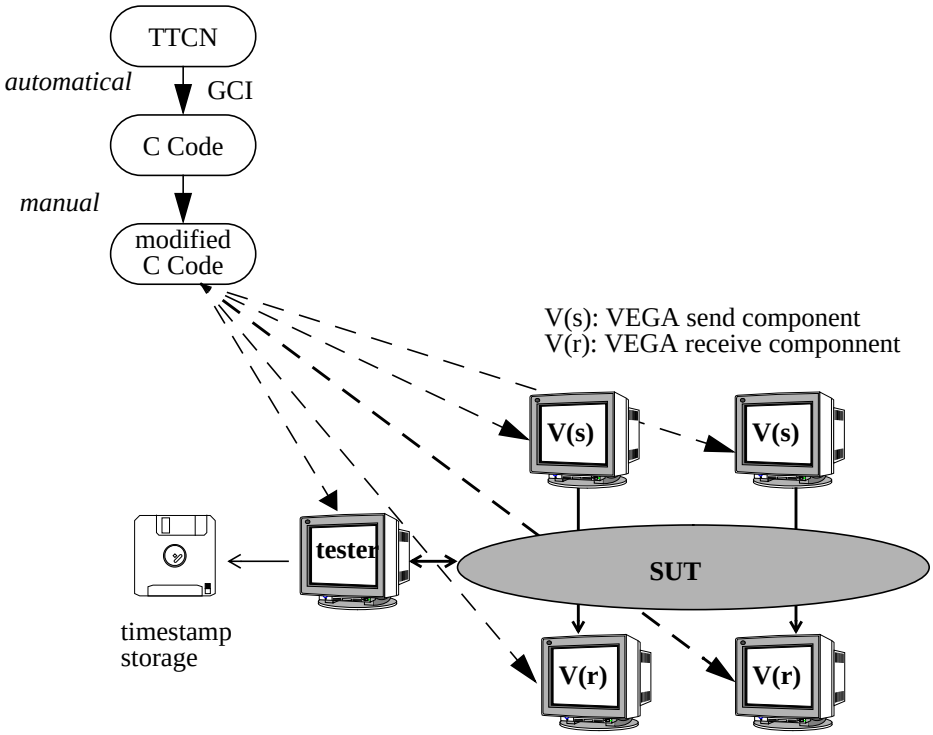


Fig 4: Technical Approach of the experiment

We have modified the original TTCN test suite to perform a measurement of the response time of a Web server to an HTTP Get operation. We have declared a measurement entity named "MeasGet" in the declaration part that measures the delay between the two events SendGet and ReceiveResults as shown in Table 9.

Measurement Declaration							
Name	**Metric**	**unit**	**event 1**	**constraint 1**	**event 2**	**constraint 2**	**Comments**
MeasGet	DELAY	ms	SendGet	SGETC	ReceiveR-esults	RRESULTC	

Table 9: HTTP measurement declaration

The repeated sampling has been implemented using a classic TTCN loop construct to make this operation more visible in this example. In this case the sampling size has been set to 10.

Test Case Dynamic Behaviour					
Test Case: www_Get **Group:** **Purpose:** **Configuration:** **Default:** **Comments:**					
Nr	**Label**	**Behaviour Description**	**Constraints Ref**	**Verdicts**	**Comments**
1	Top	N ! Connect (NumTimes := 0)	CONNECTC		

Table 10: Performance test case for the HTTP example

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:					

Table 10: Performance test case for the HTTP example

The location of the points of measurements of entity MeasGet in the dynamic behaviour are revealed in the comments column 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.

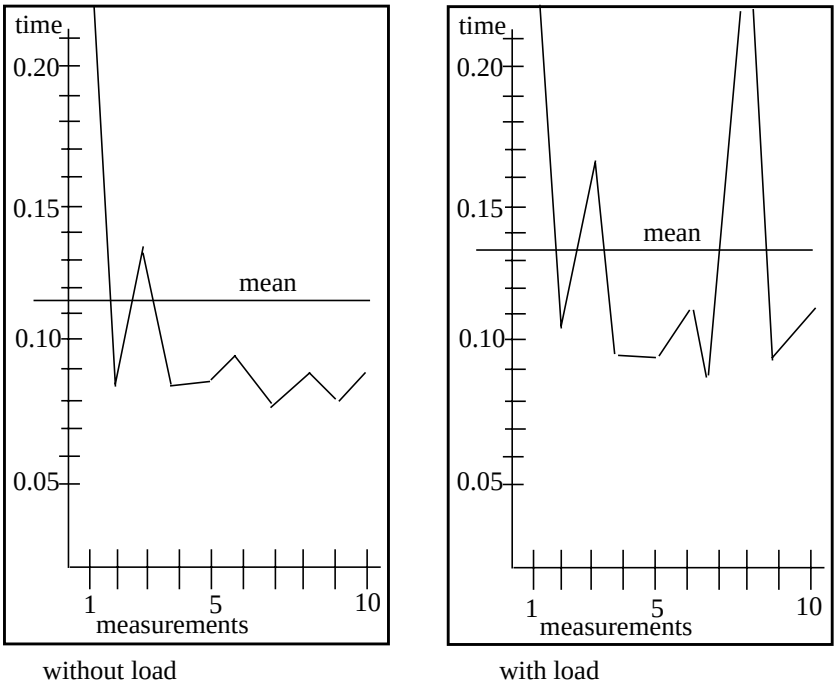
The main program of the HTTP performance test is shown in Figure 5. The GCI TTCN code of the performance test case is initiated in Line 2. Line 3 instantiates a measurement entity to collect time stamps. The co-working between TTCN GCI and VEGA is initiated by vegaTtcnBridge (Line 4). Models for background 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[3].

3. 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(); ... }
```

Fig. 5: Performance test configuration for an end-to-end service



without load with load

Fig. 6: Performance test result of the HTTP example

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. Therefore, the concrete syntax in 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 to be demonstrated on a more complex example such as he definition of a performance test suite for the ATM Adaptation Layer 5 (AAL5) that we are currently working on.

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 operational semantics for PerfTTCN will be defined next.

References

[ATM96] The ATM Forum Technical Committee: Introduction to ATM Forum Performance Benchmarking Specifications, af-test-00, May, 1996.

[CTMF] ISO/IEC 9646-1 „Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General Concepts“, 1991.

[GCI] Telelogic: ITEX 3.1 User Manual. Dec. 1996.

[JAIN91] Raj Jain: The Art of Computer Systems Performance Analysis, John Wiley & Sons, Inc. Publisher, 1996.

[ONV94] Raif O. Onvural: Asynchronous Transfer Mode Networks: Performance Issues. Artech House Inc., 1994.

[PCF9646] Keith G Knightson: OSI Protocol Conformance Testing, IS9646 explained

[TTCN] ISO/IEC 9646-3 „Information Technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The tree and tabular combined notation“, 1991.

[VEGA96] Peter Kanzow: Konzepte fur Generatoren zur Erzeugung von Verkehrslasten bei ATM-Netzen. MSc-thesis, Technical University Berlin, 1994 (german only).

[WAL97] Thomas Walter and Jens Grabowski: A Proposal for a Real-Time Extension of TTCN, KIVS’97.