A Formal Approach to Protocol Performance Testing

XU Mingwei (徐明伟) and WU Jianping (吴建平)

Department of Computer Science, Tsinghua University, Beijing 100084, P.R. China

E-mail: xmw@csnet1.cs.tsinghua.edu.cn; jianping@cernet.edu.cn

Received August 25, 1997; revised December 11, 1997.

Abstract This paper proposes a formal approach to protocol performance testing based on the extended concurrent TTCN. To meet the needs of protocol performance testing, concurrent TTCN is extended, and the extended concurrent TTCN's operational semantics is defined in terms of Input-Output Labeled Transition System. An architecture design of protocol performance test system is described, and an example of test cases and its test result are given.

Keywords protocol performance testing, extended concurrent TTCN, operational semantics, IOLTS

1 Introduction

Protocol testing is an important means to ensure the interconnection and interoperation between protocol products from different vendors. Current test activities for protocols can be classified into three classes according to their test purpose: conformance testing, interoperability testing and performance testing[1]. Conformance testing and interoperability testing are functional test, and performance testing, however, is different from them. Its purpose is to test the characteristic parameters of protocol implementation, such as packet transfer delay and throughput, so as to evaluate the efficiency of protocol implementation.

Protocol performance testing is a complex test procedure, needing several test components to coordinate and run test cases in parallel, which can be specified by concurrent TTCN. Concurrent TTCN allows more than one active test component to participate in the execution of a test case. All test components run in parallel and coordinate their behavior by exchanging coordination messages. The advantage of concurrent TTCN is that the description of test cases for complex test environment becomes easier[2].

However, concurrent TTCN cannot satisfy all needs of protocol performance testing. In protocol performance testing, it is necessary to obtain the accurate time when a test event just starts or stops. So the sequential operation of a test event and its corresponding operation, reading time, cannot be interrupted by other processes. Moreover, the traffic operation and timer operation should be extended in concurrent TTCN.

The aim of this paper is to discuss a formal approach to protocol performance testing based on the extended concurrent TTCN. To meet the needs of protocol performance testing, we extend concurrent TTCN, and formally define operational semantics for the extended concurrent TTCN in terms of Input-Output Labeled Transition System (IOLTS). Moreover, we describe a protocol performance test system based on the extended concurrent TTCN's operational semantics, and give a test case written in the extended concurrent TTCN.

The remainder of the paper proceeds as follows. Section 2 summarizes the major features of concurrent TTCN, and extends it to meet the needs of protocol performance testing. Section 3 formally defines the operational semantics of the extended concurrent TTCN. In Section 4, we describe the design of a protocol performance test system based on the

This research is supported by National Natural Science Foundation of China under Grant No. 69682002.

extended concurrent TTCN and give an example of test cases and its test result. Finally, we give some conclusions in Section 5.

2 Extended Concurrent TTCN

This section gives a short introduction to concurrent TTCN, and then extends concurrent TTCN to meet the needs of protocol performance testing. Also, a conceptual model for the extended concurrent TTCN is elaborated, whose semantic representation is discussed in Section 3.

2.1 Concurrent TTCN

The Tree and Tabular Combined Notation (TTCN)[3] is recommended by ISO to describe abstract test suite. Concurrent TTCN is an extension of TTCN. The concern of TTCN is a single test component executing a test case. Concurrent TTCN, however, allows to execute a test case by several test components (TC) running in parallel. A conceptual model of test components is depicted in Fig.1. A tester consists of exactly one main test component (MTC) and any number of parallel test components (PTCs). TCs are linked by coordination points (CPs) capable of conveying coordination messages (CMs). Communication of TCs with the environment, such as the $(N-1)$ service provider or the implementation under test (IUT), takes place at points of control and observation (PCOs).

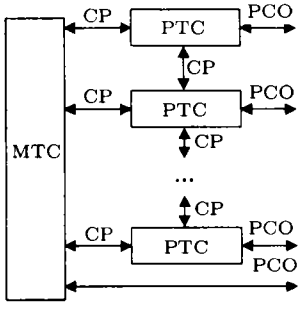

Fig.1. Conceptual model of test components.

Execution of a test case starts with the execution of MTC. It is the concern of MTC to set up all PTCs, to manage all PCOs and CPs to be connected to, and to compute the final verdict. PTCs can be created by MTC on demand. A 'create' operation associates a PTC with a behavior tree. The newly created PTC starts execution of its assigned behavior tree concurrently with MTC. MTC may explicitly terminate a PTC by executing a 'terminate' operation.

2.2 Extentions to Concurrent TTCN

Concurrent TTCN allows to execute a test case by several test components running in parallel. This is important to protocol performance testing. There are, however, still some requirements in protocol performance testing, e.g. atomic operation, traffic operation and some timer operation, that cannot be provided by concurrent TTCN. According to the needs of protocol performance testing, we extend concurrent TTCN as follows.

2.2.1 Atomic Operation

In protocol performance testing, some test cases about time parameters, such as delay, are indispensable. We can use TTCN to specify the test case, for example,

```
!a
    readtimer (t1)
      ?b
        readtimer (t2)
          (Delay:= t2 − t1)
```

There are some problems in the above specification. Since the UNIX we use is a multi-process operating system, the sequential composition of $!a$ and *readtimer* ($t1$) (*?b* and *readtimer* ($t2$)) may be interrupted by other processes. There may be time gap between $!a$ and *readtimer* ($t1$) (*?b* and *readtimer* ($t2$)), and the time gap is non-deterministic. Hence the calculated delay 'Delay := $t2 - t1$' is not accurate. To solve the problems, atomic operation is defined.

Definition 2.1. *Atomic operation can be denoted as*

$$A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_n$$

where $A_1, A_2, \ldots, A_n$ *are events and it is*

- *successfully completed if* A_{i+1} *happens immediately after the successful completion of* A_i, $(i = 1, 2, \ldots, n-1)$; *or*
- *successfully matched in a TTCN behavior tree if the first event* A_1 *is successfully matched.*

2.2.2 Traffic Operation

To test a network product's performance under more realistic assumptions, traffic generator and traffic monitor are added in the test system. Traffic generator generates a special kind of data flow in accordance with the requirement of test cases, and sends it to IUT through PCO. Traffic monitor analyzes the data flow it received from IUT, and records the result of the analysis. We define two traffic operators, *generate* and *monitor*, so that we can describe the behavior of traffic generator and traffic monitor distinctly.

Definition 2.2. *The operations of generating and monitoring traffic can be defined as:*
generate (pcoid, trid$(a_1, \cdots, a_q)$*), and*
monitor (pcoid, trid$(a_1, \cdots, a_q)$*)*
pcoid is the identifier of the PCO through which the traffic is sent or monitored, trid is the traffic source model identifier, and a_i *is the parameter of the traffic source model.*

2.2.3 Timer Operation

Four timer operators have been defined in TTCN: *start*, *cancel*, *readtimer* and *timeout*. But to meet the needs of protocol performance testing, we still need to extend timer operation.

When test components are located in different systems, timers in different systems need to be synchronized for accurately testing time parameters. One timer is selected as reference timer so that other timers can be calibrated by synchronizing operation.

Definition 2.3. *The synchronizing timer operation is defined as: synctimer (testerid).*
testerid is the identifier of a tester. This operation synchronizes the timer of a remote tester with the timer of the tester executing this command.

2.3 Conceptual Model for the Extended Concurrent TTCN

Fig.2 gives a conceptual model of a protocol performance test system. The test system defines the highest level of abstraction, which is composed of a test module, a timer and PCOs. The test module (TM) consists of all test components, traffic generator and traffic monitor, which are interconnected by CPs. A test component is a virtual TTCN machine that can perform the evaluation of test behavior expressions.

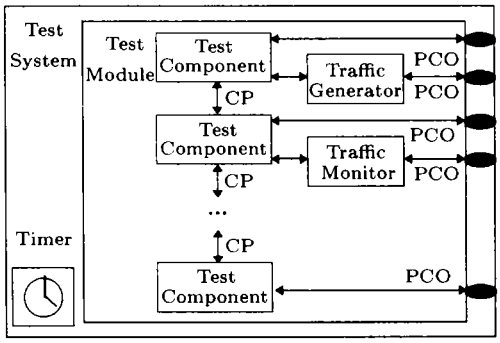

Fig.2. Conceptual model of a protocol performance test system.

3 Operational Semantics for the Extended Concurrent TTCN

This section describes our approach of defining the operational semantics of the extended concurrent TTCN. The operational semantics is defined in terms of Input-Output Labeled Transition System.

3.1 Preliminaries

Labeled Transition System (LTS) is a basic mathematical tool for modeling the behavior of systems or processes. It is widely used in protocol specifications, implementations, and tests. It also serves as semantic model for various formal specification languages, e.g. LOTOS, CSP and CCS. First we introduce the basic definition of LTS.

Definition 3.1. *A Labeled Transition System (LTS) is a 4-tuple* $\langle S, L, T, s_0 \rangle$, *where:*

1) S *is a countable, non-empty set of states;*
2) L *is a countable set of observable events;*
3) $T \subseteq S \times (L \cup \{\tau\}) \times S$ *is a set of transitions, where* τ *denotes unobservable event; element* (s, u, s') *in* T *can also be written as:* $s \xrightarrow{u} s'$, *where* $s, s' \in S$, $u \in L \cup \{\tau\}$;
4) $s_0 \in S$ *is the initial state.*

In LTS model, events in L are treated in the same way no matter what they mean. This is not the case when we want to describe the external behavior of a system communicating with others. In this case, we must distinguish input events from output events. In order to model this kind of system, a kind of LTS called Input-Output Labeled Transition System[4] is introduced.

Definition 3.2. *An Input-Output Labeled Transition System (IOLTS) p is a labeled transition system in which the set of events L is partitioned into input events* L_i *and output events* L_o, *where elements in* L_i *are events accepted by this LTS from its environment, and elements in* L_o *are events sent to its environment by this LTS. IOLTS p satisfies:*

$$p \in LTSs(L),\ L = L_i \cup L_0 \text{ and } L_i \cap L_0 = \emptyset$$

3.2 Algebraic Form of the Extended Concurrent TTCN

A definition of the operational semantics of the extended concurrent TTCN is the basis of designing a protocol performance test system based on the extended concurrent TTCN. In the test case specified by the extended concurrent TTCN, the behavior of each test component is expressed with a behavior tree in the test case. Behavior tree is a tree-like presentation of the temporal relations between test events. In order to get formal definition

of the semantics, we give the Test Behavior Expression (TBE) defined below to express the behavior of test components in algebraic form.

Definition 3.3. *The syntax of a Test Behavior Expression (TBE) is defined as follows:*

$$B =_{def} \text{stop}|\text{exit}|id?a; B|id!a; B|B\Box B|[q]; B|(I := val); B|B \gg B|B \Rightarrow B|start(tid\ val); B|cancel(tid); B|$$
$$timeout(tid); B|readtimer(t); B|synctimer(testerid); B|create(tcid, tpid[pc_1, \ldots, pc_p](a_1, \ldots, a_q)); B|$$
$$terminate(tcid); B|generate(pcoid, trid(a_1, \ldots, a_q)); B|monitor(pcoid, trid(a_1, \ldots, a_q)); B$$

3.3 Operational Semantics for the Extended Concurrent TTCN

A test component (TC) is a virtual machine that can perform the evaluation of test behavior expressions. The state of a TC can be represented by a triple ($ctrl$, sto, env), in which

1) Control part: $ctrl$ is a TBE specifying the behavior of this TC, $ctrl = Texpr(t) \in TBEs$, t denotes test case;

2) Storage part: $sto \in \{(i,v)|(i,v) \in STO = IDENT \times VALUE\}$, where $IDENT$ is the set of identifiers, and $VALUE$ is the set of values; elements in STO are pairs of identifier and its corresponding value, they represent variables and constants in test suite and their values;

3) Environment part: $env \in \{(qid,i,o) \mid (qid,i,o) \in ENV = ID \times \{\text{Input}^*\} \times \{\text{Output}^*\}\}$. The interaction of a TC and its environment is realized through many interacting points. Each interacting point is described by an identifier and a pair of input and output queues. $ID = \{pcoid\} \cup \{cpid\} \cup \{TIMER\}$, where *pcoid* represents point of control and observation, *cpid* denotes coordination point, $TIMER$ is the queue name for timer. Elements in each queue are PDUs, ASPs or CMs. Input, Output $\in$ {PDUs} $\cup$ {ASPs} $\cup$ {CMs}.

Definition 3.4. *The operational semantics of a test component is defined by the Input-Output Labeled Transition System as* $\langle S_{TC}, L_{TC}, T_{TC}, s_{0(TC)} \rangle$, *where*

1) $S_{TC} = (\{(ctrl, sto, env)\} \cup \{\emptyset_{TC}\}) \subseteq TBEs \times STO \times ENV$ *contains all possible states of the TC,* $\emptyset_{TC}$ *denotes the undefined TC; a TC becomes undefined before it is initiated or after it has stopped execution;*

2) $L'_{TC} = L_i \cup L_o \cup CREATE \cup TERMINATE \cup GENERATE \cup MONITOR \cup \{\tau\}$ *is the set of all possible events;* L_i *is the set of events input(msg), while Lo is the set of events output(pcid, msg) to PCO or CP identified by pcid for every message msg. CREATE is the set of events* create, *and TERMINATE is the set of events* terminate. *GENERATE is the set of events* generate, *and MONITOR is the set of events* monitor; τ *is an internal event;*

3) $s_{0(TC)} = (B_t, sto_0, env_0)$, $B_t = Texpr(t)$, sto_0 *and* env_0 *are the initial states of storage part and environment part respectively;*

4) $T_{TC} \subseteq S_{TC} \times L'_{TC} \times S_{TC}$.

4 Performance Testing Based on the Extended Concurrent TTCN

In this section, the architecture design of a protocol performance test system based on the extended concurrent TTCN is described, and a test case written in the extended concurrent TTCN is given.

4.1 Architecture Design of Performance Test System

Fig.3 gives the architecture of a protocol performance test system based on the extended concurrent TTCN. A protocol performance test system consists of several test components, that are traffic generators and traffic monitors, a timer and a database of test context and evaluation tree.

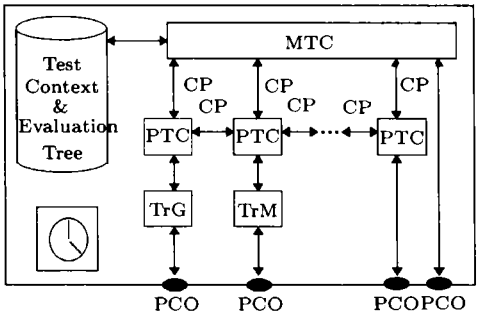

Fig.3. Architecture of a protocol performance test system.

Test components communicate with environment or with each other through interface queues and perform the evaluation of a TBE. The evaluation process is based on the operational semantics of the extended concurrent TTCN. Only one of TCs is MTC which coordinates the actions of all TCs. Traffic generatorsgenerate special kinds of traffic according to the requirement of TCs. Traffic monitors analyze the traffic received from PCOs and record the result of the analysis. Test context is the realization of the storage part of TCs, and evaluation tree is the implementation of the control part.

4.2 An Example of Test Cases

The dynamic part of a test case, whose purpose is to test the throughput of routers, is depicted in Table 1. After creating a PTC whose behavior is B, MTC sends data flow with constant bit rate to PTC through the router under testing. MTC modulates bit rate until PTC can receive all packets. PTC's behavior B is given in Table 2, and the test result of one port of Cisco 4700 with different length packets is depicted in Fig.4.

Table 1. Dynamic Part of a Test Case about Throughput

Nr	Label	Behavior Description
	L1	create (PTC, B) (dltime: =DELAY) start TM_01 readtimer TM_01 ($t1$) ⇒ generate (MT, CONT (TOTAL, dltime, data)) ⇒ readtimer TM_01 ($t2$) M? RECV_PDU (sock, recv, len, name, size) [recv<TOTAL] (dltime:=delay (dltimes)) → L1 (p:=TOTAL * PACK_SIZE*8/($t2 - t1$)) terminate (PTC)

Table 2. Dynamic Part of PTC

Nr	Label	Behavior Description
	L1	monitor (PT, CONT (TOTAL, data, len)) p! SEND_PDU (sock, recv, len, name, size) → L1

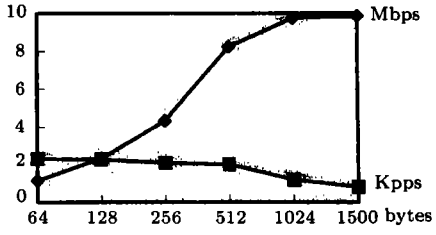

Fig.4. Throughput of one port of Cisco 4700.

5 Conclusions

In this paper, we extended concurrent TTCN to meet the needs of protocol performance testing, and then defined the operational semantics of the extended concurrent TTCN based on an extension of LTS, Input-Output Labeled Transition System. A practical architecture design of a protocol performance test system was proposed, and a test case and its test result were given.

References

[1] Hao Ruibing, Wu Jianping. Toward formal TTCN-based test execution. In *Proceedings of IEEE INFOCOM'97,* 1997.
[2] Thomas Walter, Bernhard Plattner. An operational semantics for concurrent TTCN. In *Proceedings of IFIP IWTCS V,* 1992.
[3] Conformance Testing Methodology and Framework Part 3 — The Tree and Tabular Combined Notation. ISO, November, 1991.
[4] Tretmans Jan. Testing Labeled Transition Systems with Inputs and Outputs. In *Proceedings of IFIP IWTCS VIII,* 1995.

XU Mingwei received the B.A. and Ph.D. degrees in computer science from Tsinghua University. His research interests include computer network, protocol testing and ATM.

WU Jianping is a Professor of Dept. of Computer Science, Tsinghua University, and Director of China Education & Research Network (CERNET). His research interests include computer network, protocol engineering and distributed system.