An Extension to Concurrent TTCN*

Mingwei Xu and Jianping Wu

Department of Computer Science, Tsinghua University, Beijing, 100084, China

Email: xmw@csnet1.cs.tsinghua.edu.cn and jianping@cernet.edu.cn

Abstract

This paper proposes an extension to the concurrent TTCN to meet the needs of protocol performance testing. The operational semantics of the extended concurrent TTCN are defined in terms of input-output labeled transition system. Based on the extended concurrent TTCN, the architecture of protocol performance test system is designed, and an example of test cases about throughput is given.

Keywords:

Protocol performance testing, extended concurrent TTCN, FDT, IOLTS

1. INTRODUCTION

The descriptive and formal specification of the behavior of reactive information processing system is of high relevance in many technical applications. For these systems often a complex behavior is required which includes a close co-operation between the system and its environment [1]. One of the typical application areas is protocol testing, which is an important means to ensure the interconnectivity and interoperability 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 [2]. Conformance testing and interoperability testing are functional test. Performance testing, however, is different from the above two. Its purpose is to test the characteristic parameters of protocol implementations, such as packet transfer delay and throughput, so as to evaluate the efficiency of protocol implementations.

How to describe protocol performance testing? Protocol performance testing is a complex test activity, needing several test components to coordinate and run test cases in parallel. Verbal descriptions tend to be lengthy, incomplete, to contain phrase that may be misinterpreted, and to be not well structured. Moreover, they do not follow any description standards. Therefore formal description technique (FDT) is a better way to describe protocol performance testing.

The concurrent TTCN, recommended by ISO to describe protocol abstract test suite, allows more than one active test components 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 the concurrent TTCN is that the description of test cases for complex test environment becomes easier [3].

The Concurrent TTCN, however, can not satisfy all needs of protocol performance testing, in which 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 reading time can not be interrupted by other processes. Moreover, the traffic operation and timer operation should be extended in the concurrent TTCN.

The aim of this paper is to discuss an extension to the concurrent TTCN to meet the needs of protocol performance testing. We 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, and give an example of test cases written in the extended concurrent TTCN.

The remainder of the paper proceeds as follows. Section 2 summaries the major features of the 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 extended concurrent TTCN and give an example of test cases about throughput. Finally, we draw some conclusions.

* This research is supported by National Natural Science Foundation of China under Grant No. 69473011.

2. EXTENDED CONCURRENT TTCN

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

2.1 Concurrent TTCN

The Tree and Tabular Combined Notation (TTCN) [4] is recommended by ISO to describe abstract test suite. The concurrent TTCN is an extension of TTCN. The concern of TTCN is a single test component executing a test case. The concurrent TTCN, however, allows the test system to execute a test case by several test components (TCs) running in parallel. A conceptual model of test components is depicted in Figure 2.1. 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 to convey 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).

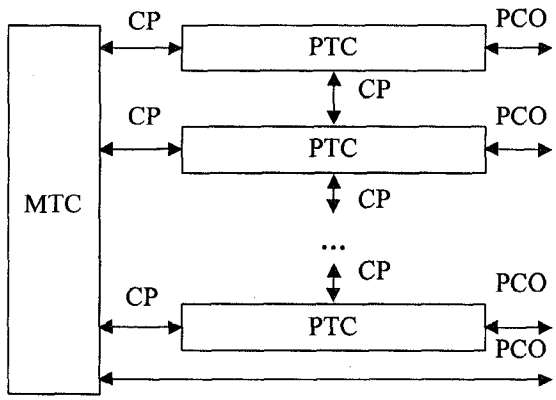

Figure 2.1 Conceptual model of test components

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

The concurrent TTCN allows the test system 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, cannot be provided by the concurrent TTCN. According to the needs of protocol performance testing, we extend the 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. Therefore 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

$$\mathbf{A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_n}$$

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

- successfully completed if A_{i+1} happens immediately after the successful completion of A_i, (i = 1, 2, ···, n-1);
- 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 receives 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 operation of generating and monitoring traffic can be defined as:

generate (pcoid, trid(a_1, ···, a_q)), and

monitor (pcoid, trid(a_1, ···, 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 it is still needed to extend timer operation to meet the needs of protocol performance testing.

When test components are located in different systems, timers in different systems need to be synchronized for accurately testing time parameters. One of timers 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 of A Protocol Performance Test System

Figure 2.2 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 consists of all test components, traffic generator and traffic monitor, which are interconnected by coordination points. A test component is a virtual TTCN machine that can perform the evaluation of test behavior expressions.

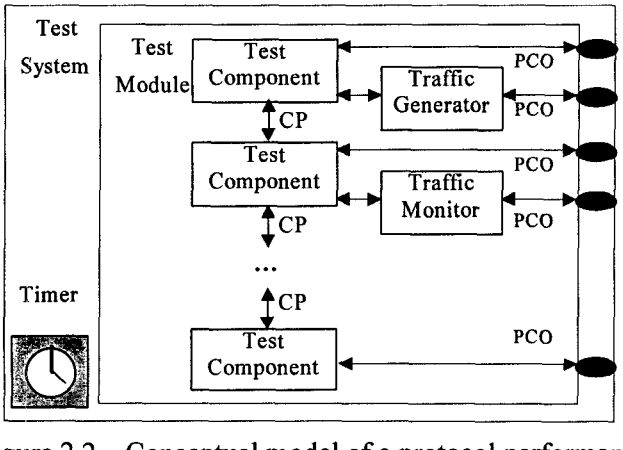

Figure 2.2 Conceptual model of a protocol performance test system

3. OPERATIONAL SEMANTICS OF THE EXTENDED CONCURRENT TTCN

The semantics of descriptive formalism have to be defined by clear mathematical means that allows an unambiguous meaning of the construct [1]. This section describes the definition of the operational semantics of the extended concurrent TTCN 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 specification, implementation, and test. 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 $<S, L, T, s_0>$, 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 an 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.

We use LTSs(L) to denote the set of all possible labeled transition systems over L. Trace is a common concept in LTS, definition 3.2 gives out its definition and some useful notation.

Definition 3.2 Let $< S, L, T, s_0 >$ be a LTS, $L' = L \cup \{\tau\}$ contains all observable and unobservable events, s, s', s_1, s_2, ... , s_n, $s_{n+1} \in S$, $u_1, u_2, ..., u_n \in L'$, let $\sigma = u_1 \cdot u_2 \cdot \ldots \cdot u_n$ be a sequence of labels in L', '·' denotes concatenation, then σ is said to be a **trace** over L'. L'^* represents the set of all possible traces over L'.

We further have the following notations:

- if $s = s1 \xrightarrow{u1} s2 \xrightarrow{u2} \cdots \xrightarrow{un} s_{n+1} = s'$, then $s \xrightarrow{\sigma} s'$.
- if $s = s1 \xrightarrow{\tau^*} s2 \xrightarrow{u} s3 \xrightarrow{\tau^*} s4 = s'$, then $s \overset{u}{\Rightarrow} s'$, $u \in L$, τ^* is the concatenation of zero or more τ.
- if $s = s1 \overset{u_1}{\Rightarrow} s2 \overset{u_2}{\Rightarrow} \cdots \overset{u_n}{\Rightarrow} s_{n+1} = s'$, then $s \overset{\sigma}{\Rightarrow} s'$.
- if $\exists s'$, $s \xrightarrow{\sigma} s'$, then $s \overset{\sigma}{\Rightarrow}$.
- if $\exists s'$, $s \overset{\sigma}{\Rightarrow} s'$, then $s \overset{\sigma}{\Rightarrow}$.

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 is introduced.

Definition 3.3 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 Li and output events Lo, where elements in Li are events accepted by this LTS from its environment, and elements in Lo are events sent to its environment by this LTS. IOLTS p satisfies:

$p \in LTSs(L)$, $L = Li \cup Lo$ and $Li \cap Lo = \varnothing$

$IOLTSs(Li, Lo)$ represents the set of all possible input-output labeled transition systems over input events Li and output events Lo. $IOLTSs(Li, Lo) \subseteq LTSs(Li \cup Lo)$

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 3.4 The syntax of a Test Behavior Expression (TBE) is defined as follows:

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

3.3 Operational Semantics of the Extended Concurrent TTCN

The operational semantics of the extended concurrent TTCN is defined in terms of hierarchical model according to the structure of the protocol performance test system conceptual model given in Figure 2.2.

3.3.1 Operational Semantics of Test Component

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 triplet (ctrl, sto, env), in which

1) Control part: *ctrl* is a TBE specifying the behavior of this TC, *ctrl* = Texpr(t) ∈ TBEs, t denotes test case;
2) Storage part: *sto* ∈ { (i,v) | (i,v) ∈ STO = IDENT × 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, which represent variables or constants in test suite and their values;
3) Environment part: *env* ∈ { (qid,i,o) | (qid,i,o) ∈ ENV = ID × {Input*} × {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} ∪ {cpid} ∪ {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 ∈ {PDUs} ∪ {ASPs} ∪ {CMs}.

Definition 3.5 The operational semantics of a test component is defined by the input-output labeled transition system as $<S_{TC}, L_{TC}, T_{TC}, s_{0(TC)}>$, where

1) $S_{TC} = (\{ (ctrl, sto, env) \} \cup \{\varnothing_{TC}\}) \subseteq TBEs \times STO \times ENV$, contains all possible states of the TC, $\varnothing_{TC}$ denotes the undefined TC. A TC becomes undefined before it is initiated or after it has stopped execution ;
2) L'_{TC} = Li ∪ Lo ∪ CREATE ∪ TERMINATE ∪ GENERATE ∪ MONITOR ∪ {τ}, is the set of all possible events. Li 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*. Finally, τ is an internal event;
3) $s_{0(TC)} = (B_t, sto_0, env_0)$, B_t=Texpr(t), sto_0, env_0 are the initial states of storage part and environment part respectively;
4) $T_{TC} \subseteq S_{TC} \times L'_{TC} \times S_{TC}$, contains transitions satisfying the inference rules defined in Appendix.

3.3.2 Operational Semantics of Traffic Generator and Traffic Monitor

A state of a traffic generator (TrG) is an element of the set

S_{TrG} = {(p, sto, env) | p ∈ TrGProc, sto ∈ STO, env ∈ ENV} ∪ $\{\varnothing_{TrG}\}$

TrGProc is the set of states of the traffic generating process. STO = {(i, v) | i ∈ IDENT, v ∈ VALUE }, ENV = {(qid,i,o) | qid ∈ ID, i ∈{Input*}, o ∈{Output*}}, $\varnothing_{TrG}$ is the undefined traffic generator.

Definition 3.6 The semantics of a traffic generator is defined by input-output labeled transition system as $<S_{TrG}$,

$L_{TrG}, T_{TrG}, s_{0(TrG)}>$, where

1) $S_{TrG} = (\{(p, sto, env)\} \cup \{\varnothing_{TrG}\}) \subseteq TrGProc \times STO \times ENV$, contains all possible states of a traffic generator;
2) $L'_{TrG} = Li \cup Lo \cup GENERATE \cup \{\tau\}$, is the set of all possible events;
3) $s_{0(TrG)} = (p, sto_0, env_0)$ is the initial state of a traffic generator;
4) $T_{TrG} \subseteq S_{TrG} \times L'_{TrG} \times S_{TrG}$, contains transitions satisfying the following inference rules:

- $(p, sto, env) \xrightarrow{output(pcid,msg)} (p', sto, env')$, if $env' = env[(pcid, i, o)/(pcid, i, o \cdot msg)]$;
- $(p, sto, env) \xrightarrow{input(msg)} (p', sto, env')$, if $env' = env[(pcid, a \cdot i, o)/(pcid, i, o)]$;
- $\varnothing_{TrG} \xrightarrow{generate(pcoid,trid(a_1,\cdots,a_q))} (p_{trid}, sto_0, env_0)$.

The operational semantics of a traffic monitor (TrM) is similar to that of a traffic generator.

Definition 3.7 The semantics of a traffic monitor is defined by input-output labeled transition system as $<S_{TrM}, L_{TrM}, T_{TrM}, s_{0(TrM)}>$, where

1) $S_{TrM}=(\{(p, sto, env)\} \cup \{\varnothing_{TrM}\}) \subseteq TrMProc \times STO \times ENV$, contains all possible states of the traffic monitor, $\varnothing_{TrM}$ is the undefined traffic monitor;
2) $L'_{TrM} = Li \cup Lo \cup MONITOR \cup \{\tau\}$, is the set of all possible events;
3) $s_{0(TrM)} = (p, sto_0, env_0)$ is the initial state of a traffic monitor;
4) $T_{TrM} \subseteq S_{TrM} \times L'_{TrM} \times S_{TrM}$, contains transitions satisfying the following inference rules:

- $(p, sto, env) \xrightarrow{output(pcid,msg)} (p', sto, env')$, if $env' = env[(pcid, i, o)/(pcid, i, o \cdot msg)]$;
- $(p, sto, env) \xrightarrow{input(msg)} (p', sto, env')$, if $env' = env[(pcid, a \cdot i, o)/(pcid, i, o)]$;
- $\varnothing_{TrM} \xrightarrow{monitor(pcoid,trid(a_1,\cdots,a_q))} (p_{trid}, sto_0, env_0)$.

3.3.3 Operational Semantics of PCOs and CPs

As stated before, PCOs and CPs are behaviorally equivalent, so it allows us to concentrate on the operational semantics of PCOs. A PCO is modeled by two queues, messages in which are managed by a PCO process.

Definition 3.8 The semantics of a PCO process can be given by input-output labeled transition system as $<S_{PCO}, L_{PCO}, T_{PCO}, s_{0(pco)}>$, where

1) $S_{PCO} = \{(p, i, o) \mid p \in PCOProc, i, o \in Msg^*\}$ contains all possible states of a PCO process. *i, o* are the sequences of messages in the input or output queue respectively. $Msg = \{PDUs\} \cup \{ASPs\}$.
2) $L'_{PCO} = Li \cup Lo \cup \{\tau\}$ is the set of all possible events;
3) $s_{0(pco)} = (p, \varnothing, \varnothing)$, $\varnothing$ represents that the input and output queues are empty at the initial state;
4) $T_{PCO} \subseteq S_{PCO} \times L'_{PCO} \times S_{PCO}$, contains transitions satisfying the following inference rules:

- $(p, i, o) \xrightarrow{input(msg)} (p, i \cdot msg, o)$;
- $(p, i, msg \cdot o) \xrightarrow{output(msg)} (p, i, o)$.

To derive the operational semantics of a CP process, it is sufficient to substitute CP for PCO in all definition, where Msg is the set of all coordination messages (CMs).

3.3.4 Operational Semantics of Test Module

A test module (TM) can be abstracted from the behavior of test components, traffic generators, traffic monitors and coordination points defined in the test module. The state of a test module is determined by the states of all TCs, TrGs, TrMs and all CP processes.

$$S_{TM} = S_{TC1} \times \cdots \times S_{TCm} \times S_{TrG1} \times \cdots \times S_{TrGp} \times S_{TrM1} \times \cdots \times S_{TrMq} \times S_{CP1} \times \cdots \times S_{CPn}$$

According to section 3.3.3

$$S_{CP} = \{(p, i, o) \mid p \in CPProc, i, o \in Msg^*\}, Msg = \{CMs\}$$

All TCs are initialized with the undefined TC $\varnothing_{TC}$ with the exception of MTC. All TrGs and TrMs are initialized with the undefined TrG $\varnothing_{TrG}$ and undefined TrM $\varnothing_{TrM}$ respectively. All CPs are initialized with empty queues. So the initial state of TM is

$$s_{0(TM)} = (s_{0(MTC)}, \varnothing_{TC1}, \cdots, \varnothing_{TCm-1}, \varnothing_{TrG1}, \cdots, \varnothing_{TrGp}, \varnothing_{TrM1}, \cdots, \varnothing_{TrMq}, s_{0(CP1)}, \cdots, s_{0(CPn)})$$

For convenience, we denote $s_{0(TM)}$ as

$$s_{0(TM)} = (s_{0(MTC)}, s_{0(CP1)}, \cdots, s_{0(CPn)})$$

The set of events a test module can perform is the set

$$L'_{TM} = Li \cup Lo \cup \{\tau\}$$

Definition 3.9 The semantics of a test module is defined by input-output labeled transition system as $<S_{TM}, L_{TM}, T_{TM}, s_{0(TM)}>$, where

1) $S_{TM} = S_{TC1} \times \cdots \times S_{TCm} \times S_{TrG1} \times \cdots \times S_{TrGp} \times S_{TrM1} \times \cdots \times S_{TrMq} \times S_{CP1} \times \cdots \times S_{CPn}$
2) $L'_{TM} = Li \cup Lo \cup \{\tau\}$ is the set of all possible events;
3) $s_{0(TM)} = (s_{0(MTC)}, s_{0(CP1)}, \cdots, s_{0(CPn)})$ is the initial state of a TM;
4) $T_{TM} \subseteq S_{TM} \times L'_{TM} \times S_{TM}$, contains transitions satisfying the inference rules defined in Appendix.

3.3.5 Operational Semantics of Test System

A test system (TS) combines the behavior of a test module and all PCO processes.

Definition 3.10 The semantics of a test system is defined by input-output labeled transition system as $<S_{TS}, L_{TS}, T_{TS}, s_{0(TS)}>$, where

1) $S_{TS} = S_{TM} \times S_{PCO1} \times \cdots \times S_{PCOn}$
2) $L'_{TS} = Li \cup Lo \cup \{\tau\}$, is the set of all possible events;
3) $s_{0(TS)} = (s_{0(TM)}, s_{0(PCO1)}, \cdots, s_{0(PCOn)})$, is the initial state of a test system;
4) $T_{TS} \subseteq S_{TS} \times L'_{TS} \times S_{TS}$, contains transitions satisfying the following inference rules:

Communication with the environment:

- $ts \xrightarrow{input(msg)} ts'$, in TS $pco_i \xrightarrow{input(msg)} pco'_i$;
- $ts \xrightarrow{output(msg)} ts'$, in TS $pco_i \xrightarrow{output(msg)} pco'_i$;

Communication between test module and PCOs:

- $ts \xrightarrow{\tau} ts'$, in TS
  $tm \xrightarrow{output(msg)} tm'$, $pco_i \xrightarrow{input(msg)} pco'_i$;
- $ts \xrightarrow{\tau} ts'$, in TS
  $tm \xrightarrow{input(msg)} tm'$, $pco_i \xrightarrow{output(msg)} pco'_i$.

4. PROTOCOL PERFORMANCE TESTING

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

4.1 Architecture Design of Protocol Performance Test System

Figure 4.1 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, traffic generators and traffic monitors, a timer and a database of test context and evaluation tree.

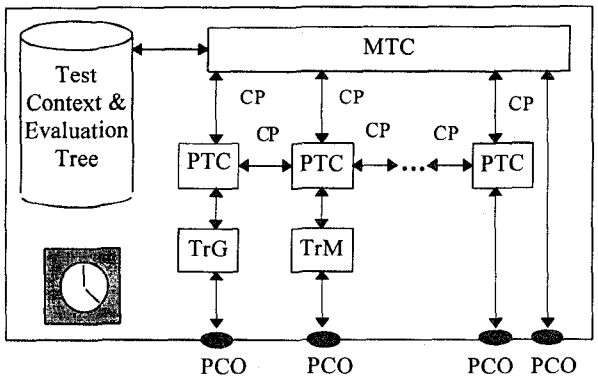

Figure 4.1 Architecture of a protocol performance test system

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 the TCs is MTC which coordinates the actions of each TC. Traffic generators generate 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

Protocol Integrated Test System (PITS) designed by Tsinghua University is an integrated tester, which can support protocol conformance testing, interoperability testing and performance testing. Now, PITS is used to test the performance of routers. The protocol performance test suite of routers is described in the extended concurrent TTCN.

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 (dltime)) —> L1 (p := TOTAL * PACK_SIZE * 8 / (t2 - t1)) terminate (PTC)

Table 4.1 Dynamic part of a test case about throughput

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

Table 4.2 Dynamic part of PTC

The dynamic part of a test case, whose purpose is to test the throughput of routers, is depicted in table 4.1. After creating a PTC whose behavior is B, MTC sends data flow with constant bit rate to PTC through the router under test. MTC adjusts bit rate until PTC can receive all packets. PTC's behavior B is described in table 4.2, and the test result of one port of Cisco 4700 with different length packets is depicted in figure 4.2.

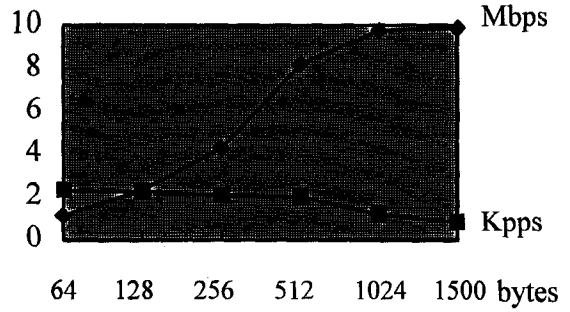

Figure 4.2 Throughput of one port of Cisco 4700

5. CONCLUSIONS

In this paper, we extended the concurrent TTCN to meet the needs of protocol performance testing, and then defined the operational semantics of the extended concurrent TTCN in terms of input-output labeled transition system. A practical architecture design of a protocol performance test system was proposed, and an example of test cases was given.

REFERENCES

1. M. Broy. Formal description techniques - how formal and descriptive are they? In: Proceedings of FORTE/PSTV'96, Kaiserslautern, Germany, 1996: 95-110.
2. Ruibing Hao and Jianping Wu. Toward formal TTCN-based test execution. In: Proceedings of IEEE INFOCOM'97, Japan, 1997.
3. Thomas Walter and Bernhard Plattner. An operational semantics for concurrent TTCN. In: Proceedings of IWPTS'92, Montreal, Canada, 1992: 101-114.
4. ISO. Conformance testing methodology and framework Part 3 - The Tree and Tabular Combined Notation, ISO, November 1991.
5. T.W.Kim, K.H.Lee and T.W.Jeong. Field trial and quality test of ATM switching system in Korea. In: Proceedings of IWPTS'96, Darmstadt, Germany, 1996: 225-236.
6. C.H.Lee and S.H.Chiu. Performance testing of distributed systems using TTCN. Dissertation, Beijing, 1996.
7. Jan Tretmans. Testing Labeled Transition Systems with Inputs and Outputs. In: Proceedings of IWPTS'95, Evry, France, 1995: 461-476.

Appendix.

1. Inference Rules of TC's Operational Semantics

- (stop, sto, env) means no transition can take place;
- $(id?a; B, sto, env) \xrightarrow{?a} (B, sto, env[(id, a \cdot i, o)/(id, i, o)])$ where $env[X/Y] =_{def} (env - \{X\}) \cup \{Y\}$;
- $(id!a; B, sto, env) \xrightarrow{!a} (B, sto, env[(id, i, o)/(id, i, o \cdot a)])$
- $(B_1 [] B_2, sto, env) \xrightarrow{\mu} (B_1', sto', env')$,
  if $(B_1, sto, env) \xrightarrow{\mu} (B_1', sto', env')$, $\mu \in L \cup \{\tau\}$;
  $(B_1 [] B_2, sto, env) \xrightarrow{\mu} (B_2', sto', env')$,
  if $(B_2, sto, env) \xrightarrow{\mu} (B_2', sto', env')$, $\mu \in L \cup \{\tau\}$;
- $([q]; B, sto, env) \xrightarrow{\tau} (B, sto, env)$, if $q = TRUE$;
  $([q]; B, sto, env) \xrightarrow{\tau} (stop, sto, env)$, if $q = FALSE$;
- $([I:=v]; B, sto, env) \xrightarrow{\tau} (B, sto[(I, x)/(I, v)], env)$, where $sto[(I, x)/(I, v)] = (sto - \{(I, x)\}) \cup \{(I, v)\}$;
- $(exit >> B_2, sto, env) \xrightarrow{\delta} (B_2, sto, env)$,
  $(B_1 >> B_2, sto, env) \xrightarrow{\mu} (B_1' >> B_2, sto', env')$,
  if $(B_1, sto, env) \xrightarrow{\mu} (B_1', sto', env')$, $\mu \in L \cup \{\tau\}$;
- $(B_1 \Rightarrow B_2; B_3, sto, env) \xrightarrow{\mu} (B_3, sto', env')$,
  if $(B_1 \Rightarrow B_2, sto, env) \xrightarrow{\mu} (exit, sto', env')$;
- $(start\ (tid\ val); B, sto, env) \xrightarrow{\tau} (B, sto, env')$,
  if $env' = env[(timer, i, o)/(timer, (tid, val) \cdot i, o)]$;
- $(cancel\ (tid); B, sto, env) \xrightarrow{\tau} (B, sto, env')$,
  if $env' = env[(timer, i, o)/(timer, (tid, 0) \cdot i, o)]$;
- $(timeout\ (tid); B, sto, env) \xrightarrow{\tau} (B, sto, env')$, if $env' = env[(timer, i, o_1 \cdot (tid, 0) \cdot o_2)/(timer, i, o_1 \cdot o_2)]$;
- $(synctimer\ (testerid); B, sto, env) \xrightarrow{\tau} (B, sto, env')$, if $env' = env[(timer, i, o)/(timer, i, o \cdot (testerid, now))]$; where *now* is the current time.
- $(readtimer\ (t); B, sto, env) \xrightarrow{\tau} (B, sto', env)$, where $sto' = sto((t, x)/(t, now))$;
- $(create(tcid, tpid[pc_1, \cdots](a_1, \cdots)); B, sto, env)_{MTC} \xrightarrow{\tau} (B, sto', env')_{MTC}$, and $\varnothing_{TC} \xrightarrow{create(tcid, tpid[pc_1, \cdots](a_1, L))} (B_{tpid}, sto, \varnothing)_{tcid}$;
- $(ter\ min\ ate(tcid); B, sto, env)_{MTC} \xrightarrow{\tau} (B, sto', env')_{MTC}$, and $(B, sto, env)_{tcid} \xrightarrow{ter\ min\ ate(tcid)} \varnothing_{TC}$.
- $(generate(pcoid, trid(a_1, \cdots)); B, sto, env) \xrightarrow{\tau} (B, sto', env')$, $(monitor(pcoid, trid(a_1, \cdots)); B, sto, env) \xrightarrow{\tau} . (B, sto', env')$.

2. Inference Rules of TM's Operational Semantics

Communication between TC and PCO:

- $tm \xrightarrow{input(msg)} tm'$, in TM $tc_i \xrightarrow{input(msg)} tc'_i$;
- $tm \xrightarrow{output(pcoid,msg)} tm'$, in TM
  $tc_i \xrightarrow{output(pcoid,msg)} tc'_i$;

Communication between TC and CP:

- $tm \xrightarrow{\tau} tm'$, in TM
  $tc_i \xrightarrow{output(cpid,msg)} tc'_i$, $cp_j \xrightarrow{input(msg)} cp'_j$;
- $tm \xrightarrow{\tau} tm'$, in TM
  $tc_i \xrightarrow{input(msg)} tc'_i$, $cp_j \xrightarrow{output(msg)} cp'_j$;

Creation of a test component:

- $tm \xrightarrow{\tau} tm'$, in TM
  $mtc \xrightarrow{create(tcid,tpid[pc_1,\cdots,pc_p](a_1,L,a_q))} mtc'$,
  $\varnothing_{TC} \xrightarrow{create(tcid,tpid[pc_1,\cdots,pc_p](a_1,L,a_q))} tc_{tcid}$;

Termination of a test component:

- $tm \xrightarrow{\tau} tm'$, in TM
  $mtc \xrightarrow{terminate(tcid)} mtc'$, $tc_{tcid} \xrightarrow{terminate(tcid)} \varnothing_{TC}$;

Generate traffic:

- $tm \xrightarrow{\tau} tm'$, in TM
  $tc \xrightarrow{generate(pcoid,trid(a_1,\cdots,a_q))} tc'$,
  $\varnothing_{TrG} \xrightarrow{generate(pcoid,trid(a_1,\cdots,a_q))} trg$;

Monitor traffic:

- $tm \xrightarrow{\tau} tm'$, in TM
  $tc \xrightarrow{monitor(pcoid,trid(a_1,\cdots,a_q))} tc'$,
  $\varnothing_{TrM} \xrightarrow{monitor(pcoid,trid(a_1,\cdots,a_q))} trm$;