Int. J. Communications, Network and System Sciences, 2011, 4, 170-179
doi:10.4236/ijcns.2011.43021 Published Online March 2011 (http://www.SciRP.org/journal/ijcns)

Verification of Session Initiation Protocol Using Timed Colored Petri Net

Safiye Kızmaz, Mürvet Kırcı
Electronics & Communication Engineering Department, Istanbul Technical University, Istanbul, Turkey
E-mail: *safiyekizmaz@gmail.com, ucerm@itu.edu.tr*
Received December 16, 2010; *revised January* 29, 2011; *accepted February* 3, 2011

Abstract

In this work, Session Initiation Protocol model is established by using Timed Colored Petri Nets (TCPN). SIP (Session Initiation Protocol) is a protocol developed to assist in providing advanced telephony services across the Internet. The Session Initiation Protocol (SIP) has become the quasi-standard for Voiceover-Internet Protocol (VoIP) communications. SIP is based on a client-server infrastructure in which user agents represent the end-terminals as clients, proxy servers handle SIP message routing between the user agents, and registrar servers store the client's contact information into a location service. By use of timed color set and useful time attributes in tokens defined in CPN tools, timer and time-related problems of SIP are modeled and analyzed. Timer is an important part for SIP, especially the INVITE transaction.

Keywords: Session Initiation Protocol, Timed Colored Petri Nets, Protocol Verification, CPN Tool

1. Introduction

Communications with Voice over IP (VoIP) have been popular because of developments of Internet. VoIP uses the Internet Protocol (IP) to transmit voice as packets over an IP network. VOIP can be achieved on any data network that uses IP, like Internet, Intranets and Local Area Networks (LAN). Before a conversation can take place between participants, protocols must be employed to establish a session, then to maintain and terminate the session. The Session Initiation Protocol (SIP) is one of the protocols being used for such purposes.

SIP is developed by the Internet Engineering Task Force (IETF) and published as Request for Comments (RFC) 3261 in 2002 [1]. Because of its increasing popularity and importance in VoIP applications [2], SIP has become a permanent element of the IP Multimedia Subsystem architecture as a signalling protocol [3]. Thus, it is important to assure that the contents of RFC 3261 are correct, unambiguous, and easy to understand. Modelling and analysing the specification using formal methods can help in achieving this goal. Moreover, from the perspective of protocol engineering, verification is also an important step of the life-cycle of protocol development [4,5], as a well-defined and verified specification will reduce the cost for implementation and maintenance.

SIP is a transaction-oriented protocol that carries out tasks through different transactions. The two main SIP transactions are the INVITE transaction for setting up a session, and the non-INVITE transaction for maintaining and closing down a session. Our current work is aimed at verifying the functional properties of the INVITE transaction.

We use Timed Colored Petri Nets (CPNs) [6] as the modeling and analyzing technique. Timed CPN have an additional property that can be used for different kinds of performance analysis. In untimed Petri Nets (Classical PNs or CPNs) it is presumed that the firing of transitions is instantaneous. Since real world actions take time to complete, tokens in timed CPN carry additional information, the time stamp. As CPNs TCPNs also have their well-developed supporting software package, the CPN Tools (Homepage of the CPN Tools).

SIP Invite Transaction is modelled and analysed with CPN in [7-10]. In [7], the functional properties of the INVITE transaction over a reliable transport medium have verified. In [8], SIP INVITE transaction was modelled and analysed when the medium is unreliable. Various forms of real-time multimedia session data such as voice, video, or text messages are carried by several protocols [11,12]. SIP works in concert with these protocols [13]. Although time factor is very important for SIP, SIP is not modelled with using TCPNs previously.

We firstly model the INVITE transaction. The rest of the paper is organized as follows. Section 2 introduces petri nets especially timed colored petri nets and definitions of timed colored petri nets are indicated. Section 3 introduces SIP INVITE transaction. Modelling and analysis of the transaction is then described in Section 4. Finally, Section 5 concludes the work and suggests future research.

2. Structure of SIP

SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions (conferences) such as Internet telephony calls. SIP is structured into four layers, each of which carries out a set of functions. The lowest layer of SIP is its syntax and encoding. Its encoding is specified using an augmented Backus-Naur Form grammar (BNF). The second layer is the transport layer. It defines how a client sends requests and receives responses and how a server receives requests and sends responses over the network. The third layer is the transaction layer with each transaction consisting of a client transaction sending requests and a server transaction responding to requests. The layer above the transaction layer is called the transaction user (TU), which creates and destroys SIP transactions, and utilises services provided by the transaction layer [1].

Among the four SIP layers, the transaction layer is the most important layer since it is responsible for request-response matching, retransmission handling with unreliable transport medium, and timeout handling when setting up or tearing down a session.

2.1. The INVITE Transaction

The INVITE client and server transactions are defined in RFC 3261 using two state machines, as shown in **Figure 1**.

1) INVITE Client Transaction

The TU communicates with the client transaction through a simple interface. When the TU wishes to initiate a new transaction, it creates a client transaction and passes it the SIP request to send and an IP address, port, and transport to which to send it. The client transaction begins execution of its state machine.

The INVITE transaction consists of a three-way handshake. The client transaction sends an INVITE, the server transaction sends responses, and the client transaction sends an ACK.

● An INVITE client transaction (**Figure 1 (a)**) has four states: ***Calling, Proceeding, Completed, and Terminated***.

● The initial state, "***calling***", must be entered when the TU initiates a new client transaction with an INVITE request.

● The client transaction must pass the request to the transport layer for transmission.

● If an unreliable transport is being used, the client transaction must start *timer A* with a value of T1. For any transport, the client transaction must start *Timer B* with a value of 64*T1 seconds. *Timer A* controls request, *Timer B* controls transaction timeouts.

● When *timer A* fires, the client transaction must retransmit the request by passing it to the transport layer, and must reset the timer with a value of 2*T1. When timer A fires 2*T1 seconds later, the request must be retransmitted again.

● If the client transaction is still in the "***Calling***" state when timer B fires, the client transaction should inform the TU that a timeout has occurred.

● The client transaction must not generate an ACK. The value of 64*T1 is equal to the amount of time required to send seven requests in the case of an unreliable transport.

● If the client transaction receives a provisional response while in the "***Calling***" state, it transitions to the "***Proceeding***" state.

● If a *Transport Err* (Error) occurs or *Timer B* expires, the client transaction moves to the ***Terminated*** state and informs its TU immediately.

● In the "***Proceeding***" state, the client transaction should not retransmit the request any longer. Furthermore, the provisional response must be passed to the TU. Any further provisional responses must be passed up to the TU while in the "***Proceeding***" state.

● When in either the "***Calling***" or "***Proceeding***" states, reception of a response with status code from 300-699 must cause the client transaction to transition to "***Completed***".

● The client transaction should start *Timer D* when it enters the "***Completed***" state, with a value of at least 32 seconds for unreliable transports, and a value of zero seconds for reliable transports. *Timer D* reflects the amount of time that the server transaction can remain in the "***Completed***" state when unreliable transports are used. This is equal to *Timer H* in the INVITE server transaction, whose default is 64*T1.

● If *Timer D* fires while the client transaction is in the "***Completed***" state, the client transaction must move to the terminated state. When in either the "***Calling***" or "***Proceeding***" states, reception of a 2xx response must cause the client transaction to enter the "***Terminated***" state, and the response must be passed up to the TU.

2.1. INVITE Server Transaction

● The server transaction is responsible for the delivery of requests to the TU and the reliable transmission of responses.

● Server transactions are created by the core when a request is received, and transaction handling is desired for that request.

● As with the client transactions, the state machine depends on whether the received request is an INVITE request. When a server transaction is constructed for a request, it enters the "***Proceeding***" state.

● The server transaction must generate a 100 (***Trying***) response unless it knows that the TU will generate a provisional or final response within 200 ms, in which case it may generate a 100 (***Trying***) response.

● If, while in the "***Proceeding***" state, the TU passes a 2xx response to the server transaction, the server transaction must pass this response to the transport layer for transmission. It is not retransmitted by the server transaction; retransmissions of 2xx responses are handled by the TU. The server transaction must then transition to the "***Terminated***" state.

● While in the "***Proceeding***" state, if the TU passes a response with status code from 300 to 699 to the server transaction, the response must be passed to the transport layer for transmission, and the state machine must enter the "***Completed***" state.

● For unreliable transports, timer G is set to fire in T1 seconds, and is not set to fire for reliable transports.

● When the "***Completed***" state is entered, timer H must be set to fire in 64*T1 seconds for all transports. Timer H determines when the server transaction abandons retransmitting the response.

● If *Timer G* fires, the response is passed to the transport layer once more for retransmission, and *Timer G* is set to fire in min (2*T1, T2) seconds. From then on, when *Timer G* fires, the response is passed to the transport again for transmission, and *Timer G* is reset with a value that doubles, unless that value exceeds T2, in which case it is reset with the value of T2.

● If an ACK is received while the server transaction is in the "***Completed***" state, the server transaction must transition to the "***Confirmed***" state. As *Timer G* is ignored in this state, any retransmissions of the response will cease.

● If *Timer H* fires while in the "***Completed***" state, it implies that the ACK was never received. In this case, the server transaction must transition to the "***Terminated***" state, and must indicate to the TU that a transaction failure has occurred.

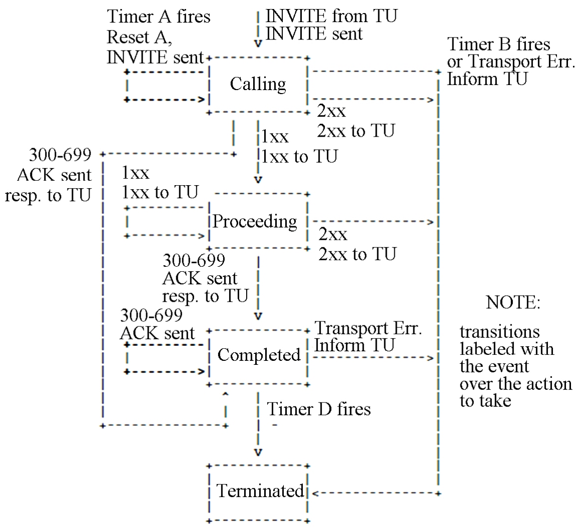
● The purpose of the "***Confirmed***" state is to absorb any additional ACK messages that arrive, triggered from retransmissions of the final response.

● Once *Timer I* fires, the server must transition to the "***Terminated***" state.

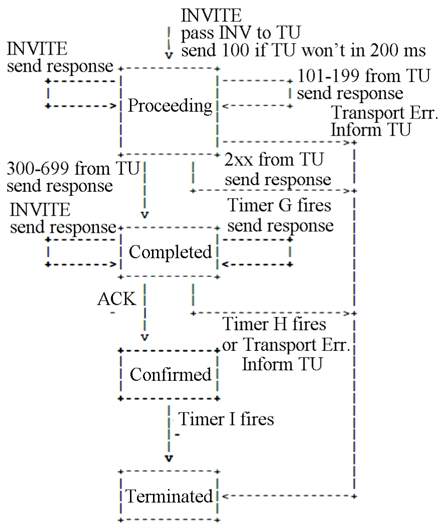
3. Petri Nets

3.1. Petri Nets Overview

Petri Nets (PNs) are a well-known formal and graphical language for modelling concurrent and asynchronous systems in the presence of conflicts, mutual exclusion, synchronization and nondeterministic aspects. In its basic form, PNs are adequate for qualitative evaluation of systems, for example to answer questions about liveness, boundedness, invariants and other characteristics of a system's model. Analysis can be performed by linear algebra techniques or by investigating the set of reachable states.



(a) INVITE client transaction



(b) INVITE server transaction

Figure 1. State machines defining SIP INVITE transaction [1].

3.2. Timed Colored Petri Nets

Most applications of CPNs are used to investigate the logical correctness of a system [14]. The CPN extended by time gives a possibility to describe the dynamic properties of a system in the time space [14-16]. The time concept of CPNs is based on the introduction of a global clock. The clock values represent the model time and they be discrete (for example integers). Each token carry a time value, also called a time stamp. The time stamp describes the earliest model time at which the token can be used, *i.e.*, removed by the occurrence of a binding element.

3.2.1. The Basic Definitions of TCPN

The original definitions of TCPN [17,18]:

Definition 1: A timed multi-set tm, over a non-empty set *S*, is a function $tm \in [\mathrm{SxR} \to \mathrm{N}]$ such that the sum:

$$tm(s) = \sum_{r \in R} tm(s,r) \tag{1}$$

is finite for all $s \in S$ The non-negative integer tm(s) is the number of appearances of the element s in the timed multi-set tm. The list:

$$tm[s] = \left[r_1,\ r_2,\ \mathbf{L},\ r_{tm(s)}\right] \tag{2}$$

is defined to contain the time values $r \in R$ for which tm(s, r) ≠ 0. Each r appears tm(s, r) times in the list, which is sorted such that $r_i \le r_{i+1}$ for all $i \in 1,\ \mathbf{L},\ tm(s)-1$.

We usually represent the time multi-set *tm* by a formal sum:

$$\sum_{s \in S} tm(s)' s @ tm[s] \tag{3}$$

By S_{TMS} we denote the set of all timed multi-sets over *S*. The non-negative integer $\{tm(s) | s \in S\}$ are called the coefficients of the timed multi-set tm, and *tm*(*s*) is called the coefficient of s. An element $s \in S$ is said to belong to the timed multi-set tm iff *tm*(*s*) ≠ 0 and we then write $s \in tm$. Each timed multi-set $tm \in S_{TMS}$ determines an ordinary multi-set $tm_u \in S_{MS}$ defined by:

$$tm_u = \sum_{s \in S} tm(s)' s \tag{4}$$

Definition 2: A timed non-hierarchical CP-nets is a tuple TCPN = (CPN, R, r_0) such that:

1) CPN= (Σ, P, TA, N, C, G, E, I) satisfying the requirements below:

a) Σ is a finite set of non empty types, called color sets.

b) P is a finite set of places.

c) T is a finite set of transitions.

d) A is a finite set of arcs such that:

$$\mathrm{P}\ \mathbf{I}\ \mathrm{T} = \mathrm{P}\ \mathbf{I}\ \mathrm{A} = \mathrm{T}\ \mathbf{I}\ \mathrm{A} = \varnothing.$$

e) N is a node function. It is defined from A into $\mathrm{P} \times \mathrm{T}\ \mathbf{U}\ \mathrm{T} \times \mathrm{P}$.

f) *C* is a colour function. It is defined from P into Σ.

g) *G* is a guard function. It is defined from *T* into expressions such that:

$$\forall t \in T : \left[Type(G(t)) = B \wedge Type(G(t)) \subseteq \Sigma\right]$$

h) E is an arc expression function. It is defined from *A* into expressions such that:

$\forall a \in A$:

$$\left[Type(E(a)) = C(p(a))_{MS} \wedge Type(VarE(a)) \subseteq \Sigma\right]$$

where *p*(*a*) is the place of *N*(*a*).

i) I is an initialization function. It is defined from P into closed expressions such that:

$$\forall p \in P : \left[(TypeI(p)) = C(p(a))_{MS}\right]$$

2) R is the set of time values, also called time stamps. It is a subset of R closed under + and containing 0.

3) r_0 is an element of R called the start time.

Definition 3: A binding of transition t is a function b defined on *Var*(*t*), such that;

1) $\forall n \in Var(t) : b(n) \in Type(n)$

2) *G*(*t*) <*b*>.

By *B*(*t*) we denote the set of all binding for *t*.

Definition 4: A binding element is a pair (*t*, *b*) where $t \in T$ and $b \in B(t)$. The set of all binding elements is denoted by BE.

Definition 5: A step Y is enabled in a state $(M_1,\ r_1)$ at time r_2 iff the following properties are satisfied:

1) $\forall r \in P : \sum_{(t,b) \in Y} E(p,t) < b > r_2 < M_l(p)$

2) $r_1 \le r_2$.

3) r_2 is the smallest element of R for which there exists a step satisfying above two restrictions.

Definition 6: When a step Y is enabled in a state $(M_1,\ r_1)$ at time r_2 it may occur, changing the state $(M_1,\ r_1)$ to another state $(M_2,\ r_2)$, where M_2 is defied by:

$$\forall r \in P : M_2(r) = \left(M_1(r) - \sum_{(t,b) \in Y} E(t,r) < b >_{r_2}\right) + \sum_{(t,b) \in Y} E(t,r) < b >_{r_2}$$

The first sum is called the removed token while the second is called the added tokens. Moreover, we say that $(M_2,\ r_2)$ is directly reachable from $(M_1,\ r_1)$ by the occurrence of the step Y at time r_2.

4. Modelling and Analysis of the SIP INVITE Transaction

4.1. Summary of Timers

The client transaction uses three timers: A, B and D. *Timer B* sets up the maximum time that the client transaction would wait in its ***Calling*** state for a provisional or final response from the server side. This timer is used no matter over what transport medium the transaction is running. *Timer A* is used only when the medium is unreliable, to control retransmissions of INVITE requests. *Timer D* also only plays its role when the medium is unreliable because its value is set to zero for reliable transport and 32 seconds for unreliable transport.

The server transaction also has three timers: G, H and I. *Timer H* is used for both reliable and unreliable medium, to set up the maximum time that the server transaction would wait in its ***Completed*** state before an ACK is received. *Timer G* is only used for unreliable transport, to control retransmissions of 300-699 responses. Timer I is set to zero seconds for a reliable transport medium, and 5 seconds for an unreliable medium.

4.2. Software Tools and TCPN

CPN Tools can simulate both basic Petri Nets and more advanced Colored Petri Nets [17]. The "color" or properties of the model are set using CPN Markup Language (CPN ML). Time is one of many properties that can be set, so timed Colored Petri Nets are available.

Since CPN Tools simulate models using Petri Nets there are only three basic elements. These are places, transitions and arcs, as shown in **Figure 2**.

The "color" modifies the way a model functions. CPN Tools can create, simulate and make state space analysis of the Petri Net model.

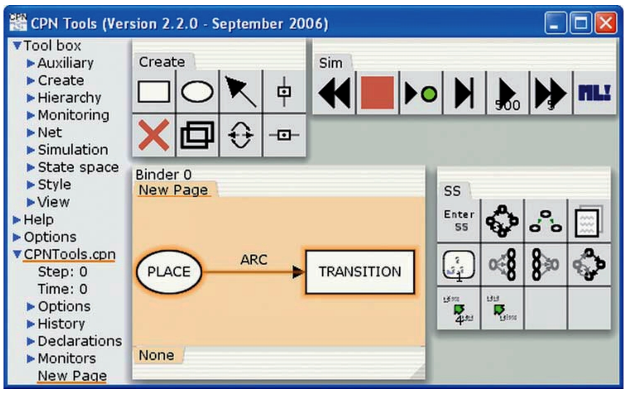


Figure 2. View of the CPN Tools with opened network window and opened menus to create, simulate and make state space analysis of the model.

Basically, Petri Net is a directed graph composed of two disjoint sets of nodes called places and transitions. Places represent states of the system, while transitions represent actions that the system can perform. To simulate an action performed by the system, appropriate transition has to "fire".

The "firing" of a transition is enabled or disabled by the tokens inside transition's input place(s). If there is an appropriate number of tokens in all input places, this depends on connecting arcs' inscriptions, then the transition is enabled and it can "fire".

4.3. TCPN Model of the INVITE Transaction

Figure 3, **Tables 1** and **2** show the TCPN model which based on state machines for the INVITE transaction. It extends the CPN model presented in [7] by adding time factor, and by modelling an unreliable transport medium. Same naming conventions are used here as in [4]. To distinguish a server transaction's state from a client transaction's state with the same name, a capitalised S is appended to the name of the state of the server transaction (except for the proceedingT state). SIP response messages (**Table 3**) are named as follows: r100 represents a 100 Trying response; r101 is for a provisional response with a status code between 101 and 199; r2xx for a 2xx response; and r3xx for a 300-699 response.

- Place *Client* is typed with colour set STATEC and its initial marking is **calling**.
- Place *Invite Sent* is typed by colour set INT.
- Transition *Send Request* is enabled only if the Client is **calling** and *Invite Sent* contains an integer 0.
- Transition *Receive Response* is enabled when a response is received and the *Client* is not terminated.
- If the client transaction receives a 300-699 response, an ACK is passed to SIP transport layer, and the *Client* changes to be completed.
- If the received response is r100, r101 or r2xx, no ACK is sent; when the response is r100 or r101, the *Client* changes to **proceeding**; and when the response is r2xx, the *Client* changes to be terminated.

- If the client transaction receives a 300-699 response, an ACK is passed to SIP transport layer, and the *Client* changes to be completed.
- If the received response is r100, r101 or r2xx, no ACK is sent; when the response is r100 or r101, the *Client* changes to **proceeding**; and when the response is r2xx, the *Client* changes to be terminated.

- Transition *Timer D* is enabled once the *Client* is completed, and its occurrence changes *Client* to be terminated.
- Transition *Client Transport Err* is enabled when the *Client* is completed. Its occurrence also changes *Client* to be terminated.

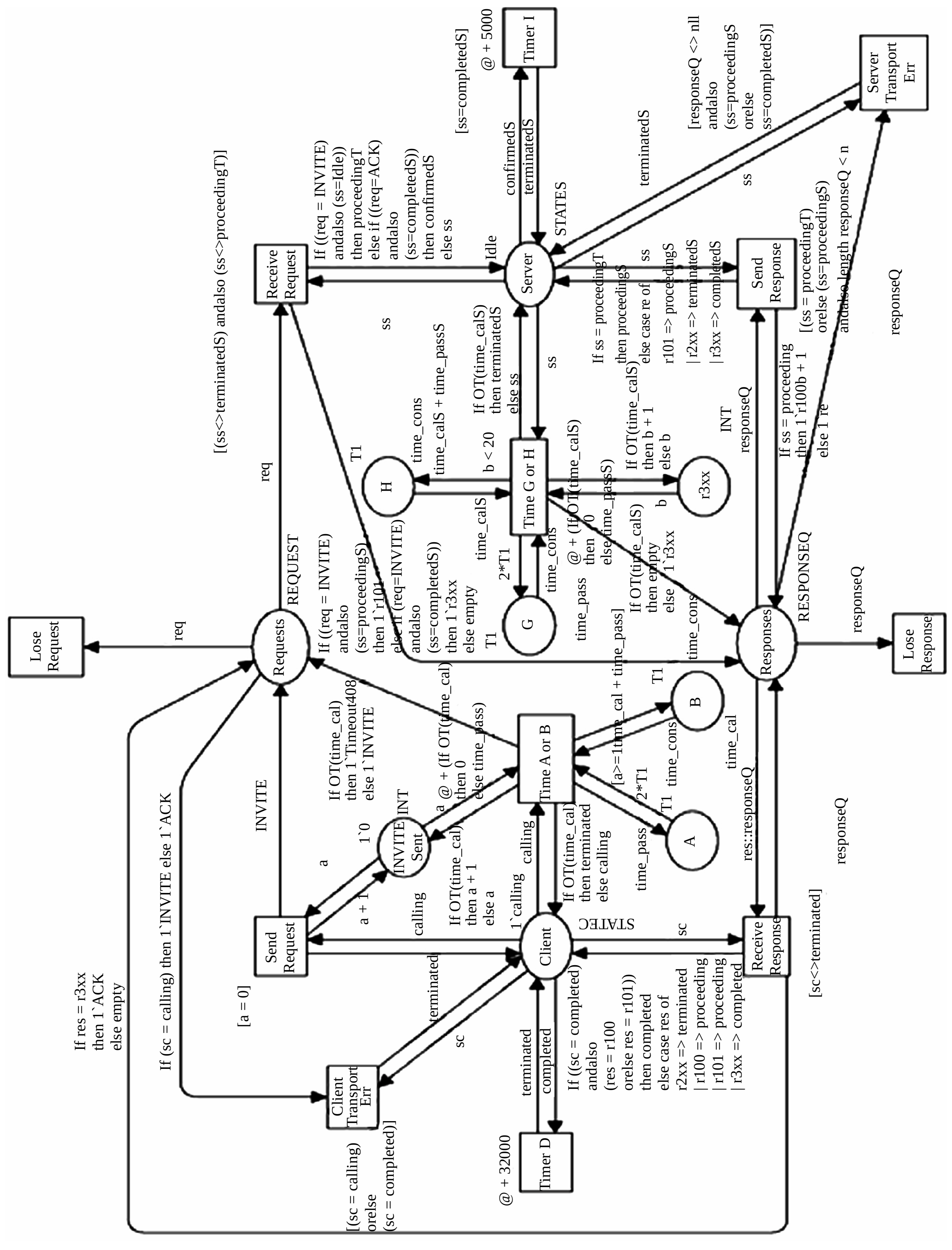

Lose Request
Requests
REQUEST
Receive Request
Server
STATES
Idle
Timer I
@ + 5000
[ss=completedS]
confirmedS
terminatedS
Server Transport Err
[responseQ <> nll andalso (ss=proceedingS orelse ss=completedS)]
Send Response
Time G or H
G
H
r3xx
Responses
RESPONSEQ
Lose Response
Time A or B
A
B
INVITE Sent
INT
Send Request
[a = 0]
Client
STATEC
Receive Response
[sc<>terminated]
Client Transport Err
Timer D
@ + 32000
[(sc = calling) orelse (sc = completed)]
If res = r3xx then 1`ACK else empty
If (sc = calling) then 1`INVITE else 1`ACK
[(ss<>terminatedS) andalso (ss<>proceedingT)]
If ((req = INVITE) andalso (ss=Idle)) then proceedingT else if ((req=ACK) andalso (ss=completedS)) then confirmedS else ss
If ((req = INVITE) andalso (ss=proceedingS) then 1`r101 else if (req=INVITE) andalso (ss=completedS)) then 1`r3xx else empty
If ss = proceedingT then proceedingS else case re of r101 => proceedingS | r2xx => terminatedS | r3xx => completedS
If ss = proceeding then 1`r100b + 1 else 1`re
[(ss = proceedingT) orelse (ss=proceedingS) andalso length responseQ < n
If OT(time_cal) then 1`Timeout408 else 1`INVITE
If ((sc = completed) andalso (res = r100 orelse res = r101)) then completed else case res of r2xx => terminated | r100 => proceeding | r101 => proceeding | r3xx => completed
res::responseQ
[a>=1time_cal + time_pass]
Figure 3. CPN model of the INVITE transaction.

Table 1. The meanings of CPN components in Figure 3.

Client Transaction	
Places	**The meanings stand for**
Client	model the states of the INVITE client transaction
Invite Sent	count the number of INVITE requests that have been transmitted and retransmitted
A	model Timer A
B	model Timer B
Transitions	**The meanings stand for**
Send Request	model how the transaction passes the original INVITE request to SIP transport layer for transmission
Receive Response	model how the client transaction receives responses and sends ACKs.
Timer D	model the ring of Timer D
Client Transport Err	control client transport error
Timer A and B	model using Timer A and B
Transport Layer	
Places	**The meanings stand for**
Requests	model the transmission of requests from the client side to the server side
Responses	model the transmission of responses in the reverse direction
Transitions	**The meanings stand for**
Lose Request	occurrences destroy requests from places Requests
Lose Response	occurrences destroy responses from places Responses
Server Transaction	
Places	**The meanings stand for**
Server	model the states of the server transaction
r3xxResent	record the number of r3xx retransmitted when Timer G fires
G	model using Timer G
H	model using Timer H
Transitions	**The meanings stand for**
Receive Request	model the reception of an INVITE or ACK request
Send Response	model how the server transaction send responses.
Server Transport Err	control server transport error
Timer I	model using Timer I
Timer G and H	model using Timer H and G

Table 2. Declarations of the CPN model.

1. colset STATEC = with calling | proceeding | completed| terminated;
2. colset STATES = with Idle | proceedingT | proceedingS | confirmedS | completedS | terminatedS;
3. colset REQUEST = with INVITE | ACK;
4. colset RESPONSE = with r100 | r101 | r2xx | r3xx;
5. colset Response = subset RESPONSE with [r101, r2xx, r3xx];
6. colset time_cons=int timed;
7. colset INT = int with 0..11; (*---variables---*)
8. var time_passS, time_calS: time_cons;
9. var time_pass, time_cal: time_cons;
10. val T1 = 500;
11. var sc : STATEC;
12. var ss : STATES;
13. var req: REQUEST;
14. var re : Response;
15. var res : RESPONSE;
16. var a,b: INT;
17. fun OT(t:time_cons):bool = if t>64*T1 then true else false;

Table 3. SIP response messages.

Timer	Value	Meaning
T1	500 ms default	RTT Estimate
T2	4 s	The maximum retransmit interval for non-INVITE requests and INVITE response
T4	5 s	Maximum duration a message will remain in the network
Timer A	initially T1	INVITE request retransmit interval, for UDP only
Timer B	64*T1	INVITE transaction timeout timer
Timer C	> 3 min	Proxy INVITE transaction timeout
Timer D	> 32 for UDP 0s for TCP/SCTP	Wait time for response retransmits
Timer E	initially T1	Non-INVITE request retransmit interval, for UDP only
Timer F	64*T1	Non-INVITE transaction timeout timer
Timer G	initially T1	INVITE response retransmit interval
Timer H	64*T1	Wait time for ACK receipt
Timer I	T4 for UDP 0s for TCP/SCTP	Wait time for ACK retransmits
Timer J	64*T1 for UDP 0s for TCP/SCTP	Wait time for non-INVITE retransmits
Timer K	T4 for UDP 0s for TCP/SCTP	Wait time for response retransmits

● When an error is reported by SIP transport layer, the ACK that has been passed to it from the transaction layer will not be sent to the server side, so when *Client Transport Err* occurs, the ACK that has been put in place *Requests* is destroyed.

● A transport error can occur when the client transaction is ***Calling***, so *Client Transport Err* is enabled as well when the *Client* is calling.

● Place *Server* is typed by colour set STATES.

● ***ProceedingT*** models the new state ***ProceedingT*** that we add to the server transaction.

● The initial marking for place *Server* cannot be proceedingT.

● Place *r3xxResent* is typed by INT.

● When the transition occurs upon receiving an INVITE request and the *Server* is Idle, a ***proceedingT*** is created in the *Server* place. In this case and only in this case, transition *Receive Request* models the operation of the TU instead of the server transaction of receiving an INVITE request from the client side

● Once the *Server* is ***proceedingT***, transition *Send Response* is enabled, thus a r100 can be put into *Responses*, and the state of *Server* is changed to ***proceedingS***.

● In the ***proceedingS*** state, *Send Response* is again enabled. When it occurs, a r101, r2xx or r3xx response is put into place *Responses*. This is represented by the variable re included in the else clause of the inscription of arc from *Send Response* to *Responses* where re is of type *Response*. Meanwhile, a ***proceedingS***, ***completedS*** or ***terminatedS*** is put in the *Server* place.

● While the *Server* is ***completedS***, if an ACK is received, the occurrence of *Receive Request* changes the *Server* to ***confirmedS***. We have assumed that the server transaction can receive INVITE or ACK requests when it is ***confirmedS***, the last else clause of the inscription of the arc from *Receive Request* to *Server* models the server transaction stays in the same state and do not send any response. The guard of *Receive Request* models that the server transaction cannot receive any requests after it is ***Terminated*** because it is destroyed by TU after the ***Terminated*** state is entered.

● If the medium is unreliable, when the *Server* is ***proceedingS*** or ***completedS***, a response (r101 or r3xx) is sent upon receiving an INVITE retransmitted by the client.

● Similar to the modelling of *Timer A* and *Timer B* requests when it is ***confirmedS***, the last else clause of the inscription of the arc from *Receive Request* to *Server* models the server transaction stays in the same state and do not send any response. The guard of *Receive Request* models that the server transaction cannot receive any requests after it is ***Terminated*** because it is destroyed by TU after the ***Terminated*** state is entered.

● If the medium is unreliable, when the Server is ***proceedingS*** or ***completedS***, a response (r101 or r3xx) is sent upon receiving an INVITE retransmitted by the client.

● When the transport medium is unreliable, we cannot use lists. Instead we let the color sets of *Requests* and *Responses* be multisets of possible requests and responses respectively, so that an occurrence of an output transition of *Requests* or *Responses* destroys a randomly picked request or response. This models that the transport medium may reorder messages, ***i.e***. messages are not received in the order they are sent (put into the *Requests* or *Responses* place).

● To model message loss, we use two transitions Lose Request and Lose Response, whose occurrences destroy requests and responses from places Requests and Responses respectively.

● Retransmission is controlled by transition “Timer A and B” with T1, an interval 2*T1 and Place A. Timeout is controlled by declared overtime function OT of 64*T1 for Timer B with transition “Timer A and B” and Place B. Time transition inscription of “Timer A and B” indicates time consuming by attaching time to “@+.”

● When timer D fires while the client transaction is in the “Completed” state, the client transaction must move to the terminated state.

● As Timer D, when timer I fires, the server must transition to the “Terminated” state.

4.4. State Space Analysis of the INVITE Transaction CPN Model

CPN tools offers specific tools to analyze properties of modeled net, such as boundness and liveness properties shown in the simulation report, automated state space calculation, supported query functions of CPN ML, simulation, and performance tools, etc.

As state space is calculated, CPN ML query functions can be utilized for further analysis. CPN ML language is used for declarations and net inscriptions. The monitoring and performance tools are useful for simulation of models, which would store simulation data for further analysis. Simulating INVITE transaction model, we can see states of telecommunication systems which use SIP protocol at the certain time. Real-time systems behaviors and carrying of Multimedia session data such as voice, video, or text messages are seen obviously by adding time factor.

In this section we define state space analysis of model which is represented above. In order to avoid state explosion problem with state space analysis, we use 3 as the maximum length of the queue in place *Response,* define colset INT = int with 0···1 and limit transitions (for example Timer A and B is enabled when [a > = 1] and Timer D is enabled when b < 17). When the model is generated, we meet infinite state space and can not analyse state space. Its reason is Timer D is enabled for every value of b. If we did not limit transition Timer D, model would be infinite loop because value of b.

A deadlock is an undesired dead marking in the state space of a CPN model, and a marking is dead if no transitions are enabled in it [19]. We also expect that the INVITE transaction has no dead code. For our model, there is no dead marking. We use state space report generated by CPN tools for analyse our model properties. The report shows that a full state space with 14 nodes and 29 arcs is generated. If we use SCC graph(strongly connected components) a full state space with 14 nodes and 29 arcs would be generated. Dead Transition Instances returns a list with all those transition instances that are dead, *i.e.*, do not appear in any occurrence sequence starting from the initial marking of the state space. For our model:

model'Client_Transport_Err 1
model'Receive_Response 1
model'Timer_A_or_B 1
model'Timer_D 1
model'Timer_I 1

TIsLive determines whether the set of transition instances (specified in the list) is live, *i.e.*, whether, from each reachable marking, it is possible to find an occurrence sequence which contains one of the transition instances. For our model:

model'Timer_G_or_H 1

TIsFairness determines whether the set of transition instances (specified in the list) is impartial, fair or just.

model'Client_Transport_Err 1	*Fair*
model'Receive_Request 1	*Fair*
model'Receive_Response 1	*Fair*
model'Send_Request 1	*No Fairness*
model'Send_Response 1	*Fair*
model'Server_Transport_Err 1	*No Fairness*
model'Timer_A_or_B 1	*Fair*
model'Timer_D 1	*Fair*
model'Timer_G_or_H 1	*No Fairness*
model'Timer_I 1	*Fair*

5. Conclusions and Future Work

In this paper, we have modelled and analysed SIP INVITE transaction using timed Coloured Petri Nets. Based on the detailed analysis of SIP, Time Coloured Petri net model of the protocol is established in the paper. By discussing and analyzing the model on the basis of properties of Petri net relating to the model, analyzing reachability tree, and calculating and analyzing the invariant, the protocol was proved to be boundedness, deadlock free, liveness and conservativeness.

In the future, we would model and analyse INVITE transaction over a reliable medium and unreliable medium and find that the INVITE transaction is free of livelocks and dead codes, as in the case of a reliable medium. We have noticed that, very recently, an Internet draft (work in progress) has been published by IETF, to propose updates to the INVITE transaction state machines [19]. The proposed updates have no impacts on the behaviour of the INVITE transaction when the transport medium is reliable, which means IETF may have not been aware of the incompleteness of [19] of the specification of the INVITE transaction. On the other hand, the proposed updates may have influence on the INVITE transaction when the transport medium is unreliable. Therefore, the other possible future work can include modelling and analysing the updated version of INVITE transaction proposed in the Internet Draft [19]. In this way, the correctness of the proposed updates given in the Internet Draft [19] can be checked and confirmed.

6. References

[1] J. Rosenberg, *et al*., "RFC 3261: SIP: Session Initiation Protocol," Internet Engineering Task Force, 2002. http://www.faqs.org/rfcs/rfc3261.html

[2] R. Arora, "Voice over IP: Protocols and Standards," Student Reports, CSE of Ohio-State University, Columbus, November 1999.

[3] R. Sparks, "SIP: Basics and Beyond," *Queue*, Vol. 5, No. 2, 2007, pp. 22-33. doi:10.1145/1229899.1229909

[4] G. J. Holzmann, "Design and Validation of Computer Protocols," Prentice Hall, Englewood Cliffs, 1991.

[5] D. Sidhu, A. Chung and T. P. Blumer, "Experience with Formal Methods in Protocol Development," *ACM SIGCOMM Computer Communication Review*, Vol. 21, No. 2, 1991, pp. 81-101. doi:10.1145/122419.122425

[6] J. Wang, "Timed Petri Nets Theory: An Application," Kluwer Academic Publishers, Norwell, 1998.

[7] L. G. Ding and L. Liu, "Modelling and Analysis of the INVITE Transaction of the Session Initiation Protocol Using Coloured Petri Nets," *Proceedings of the* 29*th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency*, Xi'an, Vol. 5062, 23-27 June 2008, pp. 132-151.

[8] L. Liu, "Verification of SIP Transaction Using Coloured Petri Nets," In: B. Mans, Ed., *Proceedings of* 32*nd Aus-*

tralasian Computer Science Conference, Wellington, 19-23 January 2009, pp. 63-72.

[9] V. Gehlot and A. Hayrapetyan, "A CPN Model of a SIP-Based Dynamic Discovery Protocol for Webservices in a Mobile Environment," *Proceedings 7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Aarhus, 24-26 October 2006, pp. 1-20.

[10] G. J. Holzmann, "Design and Validation of Computer Protocols," Prentice Hall, Englewood Cliffs, 1991.

[11] Y. Peng, Z. Yuan and J. Wang, "Petri Net Model of Session Initiation Protocol and Its Verification," *Proceedings of the IEEE International Conference on Wireless Communications, Networking and Mobile Computing*, Shanghai, 21-25 September 2007, pp. 1861-1864.

[12] H. Wan, G. Su and H. Ma, "SIP for Mobile Networks and Security Model," *Proceedings of the IEEE International Conference on Wireless Communications, Networking and Mobile Computing*, Shanghai, 21-25 September 2007, pp. 1809-1812.

[13] S. Ahson and M. Ilyas, "SIP Handbook: Services, Technologies, and Security of Session Initiation Protocol," CRC Press, Boca Raton, 2009.

[14] K. Jensen, "Coloured Petri Nets," Vol. 2, Springer, New York, 1995.

[15] M. Bago, N. Peric and S. Marijan, "Modeling Wire Train Bus Communication Using Timed Colored Petri Nets," *Proceedings of SICE Annual Conference*, Tokyo, 20-22 August 2008, pp. 2905-2910. doi:10.1109/SICE.2008.4655160

[16] Y.-S. Huang, T.-H. Chung and J.-H. Lin, "A Timed Coloured Petri Net Supervisor for Urban Traffic Networks," *IMACS Multiconference on Computational Engineering in Systems Applications*, Beijing, Vol. 2, 4-6 October 2006, pp. 2151-2156.

[17] Homepage of the CPN Tools, 2009. http://wiki.daimi.au.dk/cpntools/cpntools.wiki

[18] K. Jensen, L. Kristensen and L. Wells, "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems," *International Journal on Software Tools for Technology Transfer*, Vol. 9, No. 3, 2007, pp. 213-254. doi:10.1007/s10009-007-0038-x

[19] R. Sparks, "draft-sparks-sip-invfix-00: Correct Transaction Handling for 200 Responses to Session Initiation Protocol INVITE Requests," Internet Engineering Task Force, 2007. http://tools.ietf.org/id/draft-sparks-sip-invfix-00.txt