International Journal of Industrial Engineering & Production Research June 2020 Vol. 31, No. 2: 323-338
DOI: 10.22068/ijiepr.31.2.323

RESEARCH PAPER

Design and verification of WM protocol for Electronic Commerce Transactions

K. V. Krishnam Raju

Received 18 April 2020; Revised 28 April 2020; Accepted 3 May 2020; Published online 30 June 2020

ABSTRACT

Nowadays, much of commerce is taking place in the electronic world. E-commerce transactions take place mainly in three forms: C2C (customer to customer), B2B (Business to business), and B2C (business to customer). Out of these forms, B2C type of e-commerce is the most important one. Therefore, there is a need for a secure protocol to perform the B2C type of e-commerce transactions. In the case pf B2C type, the main participating entities are customers, websites, and merchants. In this paper, the communication between website and merchant was represented by WM protocol. The design of WM protocol must consider several issues such as problem definition, services, environment, vocabulary, and message formats. The verification of WM protocol was also performed with respect to the protocol procedure rules based on linear temporal logic. The procedure rules related to the protocol were specified in process meta language. Verification was performed using SPIN model checker and the corresponding results were reported.

KEYWORDS: *E-Commerce; WM protocol; SPIN model checker; Verification; LTL properties.*

1. Introduction

Presently, a majority of businesses are active in the form of electronic transactions. This type of business is known as e-commerce. E-commerce takes several forms: B2B (Business to Business), B2C (Business to Customer), and C2C (Customer to Customer). Out of these forms, the B2C e-commerce share in the overall global GDP is around 1.61%, which is equivalent to $1.28 trillion. In India, the total revenue made via e-commerce in 2018 was $25306 million. It is estimated that the revenue will reach $62284 million by 2023. In 2018, e-commerce users from India were estimated at around 471 million in number and will rise to 657 million by 2023. These statistics convey the potentiality of e-commerce business, particularly in the B2C form in India and worldwide. In the B2C commerce, customer directly receives either products or services from sellers. The companies that sell products directly to the consumers are called B2C companies. The protocol specification must contain five different parts [1]. Therefore, each protocol specification should include the following five parts.

- The protocol services.
- The protocol execution environment.
- The messages that are used to operate the protocol called vocabulary.
- The message format used by the protocol.
- The rules that are used by the protocol for maintaining the consistency of message exchange.

Protocol definition is something similar to the language definition. Protocol definition contains a set of messages and syntax of message format. The rules that are used by protocol represent the grammar of the protocol. Service specification defines the semantics of a language. Protocol definition must satisfy the following conditions.

- The language related to the protocol must be clear.
- The language related to the protocol represents the behavior of processes that are concurrently executing.

The concurrency related to the processes creates additional problems like timing, race condition and possible deadlocks. The number of the possible ordering of events can be very high and it is very complex to verify and analyze the protocol manually.

* Corresponding author: *K. V. Krishnam Raju*
kvkraju@srkrec.ac.in

1. *Computer Science and Engineering Department, SRKR Engineering College, Bhimavaram, Andhra Pradesh, India.*

SPIN [2],[3] model checker provides an automated verification facility. It generates a state-space model for the given protocol. The verification of properties related to the model is also performed by SPIN [4]. By using formal verification technique, several protocols are verified using SPIN. The payment system in e-banking application [5] is modeled in PROMELA and verified in SPIN. The retail banking system business flow is verified using SPIN [6]. The DHCP protocol is also verified for various properties such as the absence of deadlock, livelock, and improper termination under various conditions using SPIN [7]. The internet payment system is also modeled and verified using SPIN [8]. The verification of bidding behaviors in online auctions is also performed using SPIN [9].

Verification of protocol is the process of examining the protocol according to its specification. Verification can be performed in several ways. The most successful one is based on model checking using automated tool like Casper FDR, SPIN, etc. Sen Xu, Chin-Tser Huang, Manton M. Matthews [10] analyzed security issues on the PKM protocols using Casper FDR. Verification of PKMv3 protocol in IEEE 802.16m using CasperFDR was done in [11] and several issues were reported. Verification techniques were also applied to ASK Mobile security protocol [12] and Intrusion tolerant group membership protocol [13]. WPA-PSK Authentication Protocol [14] and WPA-GPG Authentication Protocol [15] were verified using CasperFDR. Kazhamiakin, Pistore, Roveri, proposed a novel approach to the formal specification and verification of distributed processes in a web service framework [16]. Ribeiro, Fernandes, and Pinto proposed a model checking approach to embedded systems related to liveness, deadlock freedom, and structural conflicts [17]. Cryptographic protocols are also modeled using SPIN [18]. Gluck, P.R. and Holzmann, G.J described the model checking process, the methods, and conditions necessary to successfully perform model checking on the DS1 flight software [19]. SPIN model checker is also applied to verify a multi-threaded plan execution programming language [20].

2. Secure E-Commerce Protocol

Secure E-Commerce protocol is a point-to-point protocol, because the communication takes place between the two parties including customer and merchant. Using this protocol, the customer and the merchant can transfer the information related to the e-commerce transaction securely.

Service Specification of E-Commerce Protocol

The electronic commerce protocol must provide end-to-end service information exchange service between the two communicating parties.

Assumptions about the environment

The design of the protocol is based on full-duplex communication. It uses voice-grade switched telephone lines for message transfer. Another assumption about the environment is that the communication line is dedicated. Accordingly, one can hide some of the problems related to congestion control, routing, and delays. For Example, consider that the maximum distance between any two communicating nodes is around 20000 kilometers. The electric signal propagation time is around 30000KM/Sec. Therefore, the minimum time to transfer messages between the two places on earth is around 0.7 seconds.

Protocol vocabulary

Assume that protocol is a black box. The protocol provides its functionality depending on the communication with the environment. The protocol exchanges messages with the communication parties via a data link layer using internal message channels.

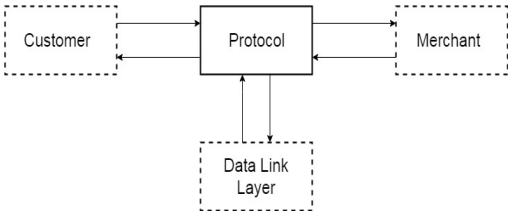

Fig. 1. Protocol Environment.

Types of messages needed for the communication

The black box accepts several types of messages from participating entities. The B2C e- commerce protocol is in the following format.

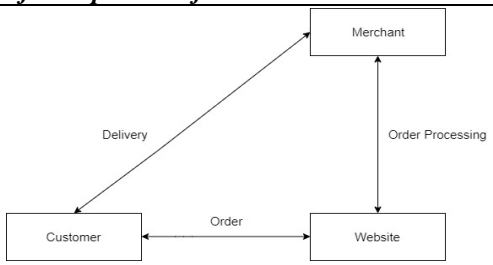

Fig. 2. Interactions of the electronic commerce protocol.

According to the above diagram, the communication is categorized into the following ways: (a) communication between the customer and the website, (b) communication between the website and the merchant, and (c) communication between the merchant and the customer. Therefore, the electronic commerce protocol is further divided into three sub protocols:
CW Protocol (Between Customer and Website)
WM Protocol (Between Website and Merchant)
MC Protocol (Between Merchant and Customer)

3. WM Protocol

WM protocol is a point-to-point communication protocol. This protocol establishes end-to-end service between the website and merchant. It uses several intermediate machines for transferring messages between the website and merchant. The service specification, environment, protocol vocabulary, message format, and procedure rules are to be considered in the WM protocol design.

Service specification of WM protocol:
The WM Protocol must provide a reliable end-to-end communication between the customer and website. Thus, services include connection establishment, data transfer, transmission error recovery, flow control, and connection termination.

The environmental assumptions of WM Protocol:
The design of the WM Protocol is based on full-duplex communication over voice-grade switched telephone dedicated line. The maximum distance between the website and merchant is 20000 kilometers. The electric signal propagation time for voice-grade switched telephone line is around 30000KM/ Sec.

Therefore, the propagation time required to transfer messages between the website and merchant is around 0.7 second.

WM protocol vocabulary:
The WM Protocol exchanges messages between two remotely located systems, the website and the merchant, via a data link layer.

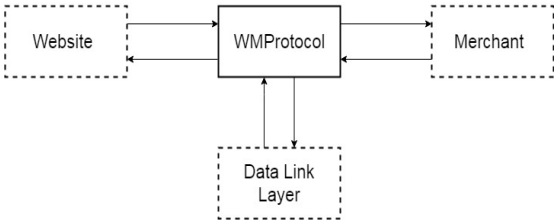

Fig. 3. WM Protocol Environment.

In the WM Protocol, the following three phases should be completed in a specific order: (a) establishment of the connection between the website and merchant, (b) data transfer between the website and merchant, and finally (c) the connection termination between the website and merchant. In the connection establishment phase, the WM sub protocol also uses the following eight messages.

connectwm: Connection establishment message from the website to the merchant.

acceptmw: Acceptance message from merchant to the website for connectwm message.

rejectmw: Rejection message from merchant to the website for connectwm message.

syncwm: Synchronization message from the website to the merchant.

sync_ackmw: Acknowledgement message from the merchant to the website for syncwm message.

syncmw: Synchronization message from the merchant to the website.

sync_ackwm: Acknowledgement message from the website to the merchant for syncwm message.
ackwm: Acknowledgement message from the website to the merchant.
During data transfer phase, eight messages are considered:
opwm: Order details from the website to the merchant.
ocwm: Order cancel message from the website to the merchant.
ormw: Order response message from the merchant to the website.
ocmw: Order cancel message from the merchant to the website.
pdwm: Payment details from the website to the merchant.
prmw: Payment response message from the merchant to the website.
adwm: Authentication details from the website to the merchant.
armw: Authentication response message from the merchant to the website.
During connection termination, only closewm message is used. Totally, seventeen messages are involved, as well.

WM Protocol message format

Each message in WM Protocol requires the following fields: type, data (optional), sequence number, and checksum. In addition, it also requires message delimiters that represent the starting and ending points of the message. They are STX and ETX. The typical message format can be assumed as follows.

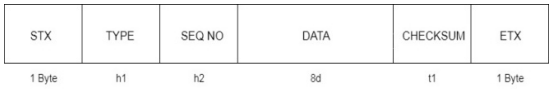

Fig. 4. WM Protocol Message Format.

In the above figure, the delimiters STX and ETX are assigned with one-byte size. Now, the size of the remaining fields type, seqno, data, and checksum must be computed. WM Protocol uses seventeen messages for its operation. Therefore, it requires 5 bits to represent fourteen messages. Therefore, h1 is equivalent to 5 bits.

$$\therefore h1 = 5\ bits. \quad (1)$$

The bandwidth of the switched telephone network is 1200 bits/second. According to the assumption about the WM protocol environment, the message transfer time is 0.7 seconds. Thus, it is long enough to transmit 840bits from a website to a merchant. Therefore, the website can transmit 1680 bits before receiving an acknowledgement message from the merchant. The number of outstanding messages for WM protocol is calculated using the following formula.

$$Number\ of\ Outstanding\ messages = 2^{(h2-1)} \quad (2)$$

If h2 is equivalent to two bits, it implies that two messages are outstanding according to selective repeat continuous ARQ flow control method. Therefore, the website can transmit a new message without receiving the previous message acknowledgement in WM protocol.

$$\therefore h2 = 2\ bits. \quad (3)$$

By considering that h2 size is equivalent to two bits, no time is lost if the message size is 1680bits. The error-free duration is around 125000 bits. Therefore, the message size ranges between 1680 and 125000 bits. The optimum size of data can be calculated as in the following. The data message overhead length is t bits. The data field length is $d = 8D$. The length of acknowledgement control message is a. Probability of data message distorted or lost in WM protocol is WMP_d. Probability of acknowledgement message distorted or lost in WM protocol is WMP_a. Now, assume that $WMP_d = WMP_a = 0$. (the absence of errors). The total bits required to transmit one message successfully in WM protocol is $(t + d + a)$ bits. The overhead of the one message transfer in WM protocol is around $(t + a)$bits. The WM protocol efficiency is calculated by $WME = \frac{d}{(d+t+a)}$.
In order to increase the WM protocol efficiency, d is as large as possible. In case of the presence of errors, assume that $WMP_d \neq 0$ and $WMP_a \neq 0$.
Probability of data message that is not distorted or lost in WM protocol is $(1 - WMP_d)$. Probability of acknowledgement message that is not distorted or lost in WM protocol is $(1 - WMP_a)$. The probability of message being retransmitted in WM protocol is WMP_r.

$\therefore WMP_r = (1-(1-WMP_d)(1-WMP_a))$. (4)

WMP_i represents the probability of i subsequent transmissions in WM protocol that can successfully transfer message from sender to receiver and is equal to $(i-1)$ retransmissions, followed by one final successful transmission of messages.

$$\therefore WMP_i = WMP_r^{(i-1)}(1-WMP_r). \quad (5)$$

WMR is used to represent the expected number of transmissions per message in WM protocol.

$$WMR = \sum_{i=1}^{\infty} i\, WMP_i. \quad (6)$$
$$WMR = \sum_{i=1}^{\infty} i\, WMP_r^{(i-1)}(1-WMP_r). \quad (7)$$
$$WMR = (1-WMP_r)\sum_{i=1}^{\infty} i\, WMP_r^{(i-1)}. \quad (8)$$
$$WMR = (1-WMP_r)\sum_{j=0}^{\infty}\ \sum_{i=j}^{\infty} WMP_r^i. \quad (9)$$
$$WMR = \frac{(1-WMP_r)\sum_{j=0}^{\infty} WMP_r^j}{1-WMP_r}. \quad (10)$$
$$WMR = 1/(1-WMP_r). \quad (11)$$

The WM protocol efficiency in case of errors is $WME = \frac{d}{WMR(d+t+a)}$. Consider the CRC approach to the checksum calculation for error control in WM protocol. The average time duration of burst error is assumed around 10 milliseconds in WM protocol. Therefore, the maximum number of bits affected by burst error is 12bits in WM protocol.

$\therefore t1 = 16\ bits = 2\ Bytes.$
Now, $h1 = 5bits$, $h2 = 2bits$, $STX = 8bits$, $ETX = 8bits$, and checksum $t1 = 16\ bits$. The data message overhead length is $t = (STX + h1 + h2 + t1 + ETX)$.
$\therefore t = (8+5+2+16+8) = 39bits.$

The acknowledgement message overhead length is $a = (STX + h1 + h2 + t1 + ETX)$.
$\therefore a = (8+5+2+16+8) = 39bits.$
$\therefore t = a = 39bits.$
According to the assumption that the error-free interval is around 125K bits, the number of WM protocol data messages that can be transmitted at an error-free interval is $(125 \times 10^3)/(d+t)$. Similarly, the number of WM acknowledge messages can be transmitted at an error-free interval is $(125 \times 10^3)/t$.

$\therefore WMP_d = (1-((125\times10^3)/(\text{d+39})))$
$WM\,P_d = (d+39)/(125\times10^3)$
$\therefore WMP_a = (1-((125\times10^3)/39))$
$WMP_a = 39/(125\times10^3)$
$WMP_a = 3.12\times10^{-4}$
$WMR = 1/(1-WMP_r)$
$WMR = 1/(1-(1-(1-WMP_d)(1-WMP_a)))$. (12)
$WMR = 1/((1-WMP_d)(1-WMP_a))$. (13)
The efficiency of the protocol in the presence of errors is
$$WME = \frac{d}{WMR(d+t+a)}. \quad (14)$$
$$\therefore WME = \frac{d}{WMR(d+39+39)}. \quad (15)$$
$$WME = \frac{d}{WMR(d+78)}. \quad (16)$$
$$WME = \frac{d\,(1-WMP_d)\,(1-WMP_a)}{(d+78)}. \quad (17)$$
$$WME = \frac{d\,(1-((d+38)/(125\times10^3)))\,(1-(3.12\times10^{-4}))}{(d+78)} \quad (18)$$

Plot the graph for the above equation to calculate the efficiency based on the length of the message in bits in WM protocol.

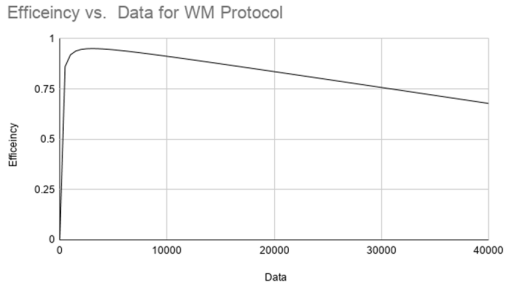

Fig. 5. WMProtocol graph for Data Vs Efficiency.

According to the above graph, the protocol maximum efficiency is around 95% when data length is around 3000 bits. This is equivalent to 375 bytes.

$\therefore d = 3000\ bits$

$\therefore d = 375\ Bytes$

The message format of the WM Protocol can be represented as follows:

Struct {
Unsigned wmtype 4;
Unsigned wmseqno 2;
Unsigned char wmdata [375];
Unsigned char wmchecksum [2];
}messagewm;

Effects of rounding the fields in WM Message Format.

Now, round every field in the message format of WMProtocol to the nearest multiple of eight.

Then $\therefore h1 = 8bits, \therefore h2 = 8bits$. The lengths of the remaining fields are the same for STX, ETX, and checksum. The total number of bits increased due to rounding to 09 bits. Now, replot the graph for WM protocol efficiency with respect to the following function:

$$\therefore WME = \frac{d}{WMR(d+48+48)}$$

$$WME = \frac{d}{WMR(d+96)}$$

$$WME = \frac{d\,(1 - WMP_d)\,(1 - WMP_a)}{(d+96)}$$

$$WME = \frac{d\,(1 - ((d+48)/(125 \times 10^3)))\,(1 - (3.84 \times 10^{-4}))}{(d+96)}$$

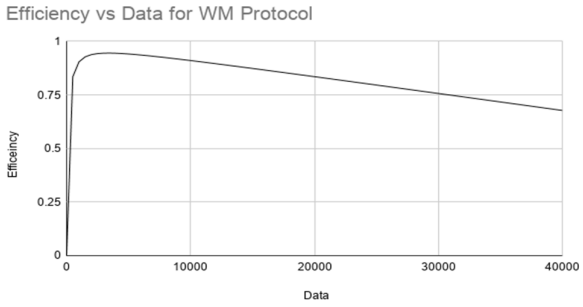

Fig. 6. WMProtocol graph for Data Vs Efficiency after increasing the overhead.

After adding 10 bits to the overhead, the efficiency is reduced by around 0.5% from 95% to 94.5%. However, the length of the data field can be increased to 3500 bits from 3000 bits to achieve 95% efficiency. This is equivalent to 438 bytes. After rounding, the message format of the WM Protocol can be modified in the following way.

Struct
{Unsigned char wmtype;
Unsigned char wmseqno;
Unsigned char wmdata [438];
Unsigned char wmchecksum [2];
}wmmessage;

The WMProtocol is divided into several layers in terms of presentation, session, and flow control layers.

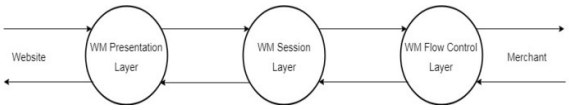

Fig. 7. WM Protocol communication layers.

Procedure Rules for WM Protocol specified in Process meta Language (PROMELA)

```
proctype wmpresent(bit n)
{byte wmstatus, wabort;
IDLE:
        if
        :: webs_to_pres[n]?opwm ->
      do
      :: wabort = 0; goto TRANSFER
      :: webs_to_pres[n]?ocwm -> skip
      od;
```

```
    fi;
        if
        :: webs_to_pres[n]?pdwm ->
      do
      :: wabort = 0; goto TRANSFER
      :: webs_to_pres[n]?ocwm -> skip /* ignore */
      od;
    fi;
        if
        :: webs_to_pres[n]?adwm ->
      do
      :: wabort = 0; goto TRANSFER
      :: webs_to_pres[n]?ocwm -> skip /* ignore */
      od;
        fi;
TRANSFER:
        if
        :: webs_to_pres[n]?opwm -> pres_to_sess[n]!opwm;
     do
     :: webs_to_pres[n]?ocwm ->
        if
        :: (!wabort) -> wabort = 1; pres_to_sess[n]!ocwm
        :: (wabort) ->  skip
        fi;
    :: sess_to_pres[n]?ormw -> goto DONE
    :: sess_to_pres[n]?ocmw(wmstatus) ->
        if
        :: (wmstatus == FATAL || wabort) -> goto FAIL
        :: (wmstatus == NON_FATAL && !wabort) -> goto TRANSFER
        fi;
    od;
        fi;
  if
        :: webs_to_pres[n]?pdwm->pres_to_sess[n]!pdwm;
     do
     :: webs_to_pres[n]?ocwm ->
        if
        :: (!wabort) -> wabort = 1; pres_to_sess[n]!ocwm
         :: (wabort) -> skip
        fi
    :: sess_to_pres[n]?prmw -> goto DONE
    :: sess_to_pres[n]?ocmw(wmstatus) ->
        if
        :: (wmstatus == FATAL || wabort) -> goto FAIL
        :: (wmstatus == NON_FATAL && !wabort) -> goto TRANSFER
        fi
    od;
 fi;
if
:: webs_to_pres[n]?adwm->pres_to_sess[n]!adwm;
     do
     :: webs_to_pres[n]?ocwm ->
        if
        :: (!wabort) -> wabort = 1; pres_to_sess[n]!ocwm
        :: (wabort) -> skip
```

```
            fi
        :: sess_to_pres[n]?armw -> goto DONE
        :: sess_to_pres[n]?ocmw(wmstatus) ->
            if
            :: (wmstatus == FATAL || wabort) -> goto FAIL
            :: (wmstatus == NON_FATAL && !wabort) -> goto TRANSFER
            fi
        od;
 fi;
DONE:
            if
            :: sess_to_pres[n]?ormw->pres_to_webs[n]!ormw; goto IDLE
            fi;
            if
            :: sess_to_pres[n]?prmw->pres_to_webs[n]!prmw; goto IDLE
            fi;
            if
            :: sess_to_pres[n]?armw->pres_to_webs[n]!armw; goto IDLE
                fi;
FAIL:
           pres_to_webs[n]!ocmw;
           goto IDLE}
proctype wmsession(bit n)
{bit toggle;
byte type,status;
IDLE:
        do
        :: pres_to_sess[n]?type ->
         if
         :: (type==(opwm||pdwm||adwm||ocwm)) -> goto DATA_OUT
         :: (type!=(opwm||pdwm||adwm||ocwm)) ->
         fi;
            :: flow_to_sess[n]?type->
         if
         :: (type==(ormw||prmw||armw||ocmw)) -> goto DATA_IN
         :: (type!=(ormw||prmw||armw||ocmw))
         fi;
            od;
DATA_OUT:
        sess_to_flow[n]!syncwm,toggle;
        do
        :: flow_to_sess[n]?sync_ackmw,type ->
                        if
                        :: (type != toggle)
                        :: (type == toggle) -> break
                        fi
        :: timeout ->
                    sess_to_pres[n]!rejectmw(FATAL);
                        goto IDLE
        od;
        toggle =  1 - toggle;
        sess_to_flow[n]!connectwm;
        if
        :: flow_to_sess[n]?acceptmw ->
           skip
```

```
        :: flow_to_sess[n]?rejectmw ->
           sess_to_pres[n]!rejectmw(FATAL);
           goto IDLE
        :: timeout->
           sess_to_pres[n]!rejectmw(FATAL);
           goto IDLE
        fi;
            do
        :: pres_to_sess[n]?opwm -> sess_to_flow[n]!opwm
        :: pres_to_sess[n]?pdwm -> sess_to_flow[n]!pdwm
        :: pres_to_sess[n]?adwm -> sess_to_flow[n]!adwm
        :: pres_to_sess[n]?ocwm -> sess_to_flow[n]!ocwm
            od;
            do
       :: pres_to_sess[n]?ocwm
       :: flow_to_sess[n]?ocmw-> sess_to_pres[n]!ocmw;
                goto IDLE
            od;

DATA_IN:
            do
       :: flow_to_sess[n]?ormw -> sess_to_pres[n]!ormw
       :: flow_to_sess[n]?prmw -> sess_to_pres[n]!prmw
       :: flow_to_sess[n]?armw -> sess_to_pres[n]!armw
       :: flow_to_sess[n]?ocmw -> sess_to_pres[n]!ocmw
            od;
          sess_to_flow[n]!ocwm;
          goto IDLE}
proctype wmflow(bit n)
{bool busywm[M];
byte qwm;
byte mwm;
byte swm;
byte windowwm;
byte typewm=(opwm||pdwm||adwm||ocwm||syncwm||sync_ackwm);
byte typemw=(ormw||prmw||armw||ocmw||syncmw||sync_ackmw);
bit receivedwm[M];
bit xwm;
byte pwm;
byte I_bufwm[M],O_bufwm[M];
do
::(windowwm < W && len(sess_to_flow[n]) > 0 && len(flow_to_dll[n])<QSZ) ->
  sess_to_flow[n]?typewm;
  if
  :: (typewm!=syncwm)->
     windowwm=windowwm+1;
     busywm[swm]=true;
     O_bufwm[swm]=typewm;
     flow_to_dll[n]!typewm,swm;
     swm=(swm+1)%M
  fi;
  if
           :: (typewm == syncwm) ->  windowwm = 0; swm = M;
           do
        :: (swm > 0) -> swm = swm-1;
```

```
        busywm[swm] = false
      :: (swm == 0) -> break
      od;
        fi;
:: dll_to_flow[n]?typemw,mwm ->
  if
  :: (typemw == sync_ackmw) -> busywm[mwm] = false
  :: (windowwm > 0 && busywm[qwm] == false) -> windowwm = windowwm - 1; qwm = (qwm+1)%M
  :: (timeout && windowwm > 0 && busywm[qwm] == true) ->
        flow_to_dll[n]!O_bufwm[qwm],qwm
  :: dll_to_flow[n]?typemw,mwm ->
        if
        :: (typemw == syncmw) -> mwm = 0;
        do
        :: (mwm < M) -> receivedwm[mwm] = 0; mwm = mwm+1
        :: (mwm == M) -> break
    od;
    flow_to_dll[n]!sync_ackwm,mwm
        :: (typemw == sync_ackmw) -> flow_to_sess[n]!sync_ackmw,mwm
        :: ( typemw!= syncmw && typemw!= sync_ackmw)->
      if
      :: (receivedwm[mwm] == true) -> xwm = ((0<pwm-mwm && pwm-mwm<=W) || (0<pwm-mwm+M
&& pwm-mwm+M<=W));
        if
        :: (xwm) -> flow_to_dll[n]!ackwm,mwm
        :: (!xwm)
        fi;
      :: (receivedwm[mwm] == false) -> I_bufwm[mwm] = typewm; receivedwm[mwm] = true;
receivedwm[(mwm-W+M)%M] = false
      fi;
        :: (receivedwm[pwm] == true && len(flow_to_sess[n])<QSZ && len(flow_to_dll[n])<QSZ) ->
      flow_to_sess[n]!I_bufwm[pwm];flow_to_dll[n]!ackwm,pwm;pwm = (pwm+1)%M
    fi;
  fi;
 od;}
```

Random execution of WM Protocol:

```
0:   proc  -  (:root:) creates proc  0 (:init:)
Starting wmpresent with pid 1
  1:  proc  0 (:init:) creates proc  1 (wmpresent)
0 :init ini run wmpresent(
Starting wmsession with pid 2
  2:  proc  0 (:init:) creates proc  2 (wmsession)
0 :init ini run wmsession(
Starting wmflow with pid 3
  3:  proc  0 (:init:) creates proc  3 (wmflow)
0 :init ini run wmflow(0)
timeout
#processes: 4
  3:  proc  3 (wmflow) kvkrwm.pml:191 (state 68)
  3:  proc  2 (wmsession) kvkrwm.pml:120 (state 13)
  3:  proc  1 (wmpresent) kvkrwm.pml:30 (state 9)
  3:  proc  0 (:init:) kvkrwm.pml:25 (state 5)
4 processes created
```

Fig. 8. Random Execution of WM Protocol in SPIN Model Checker.

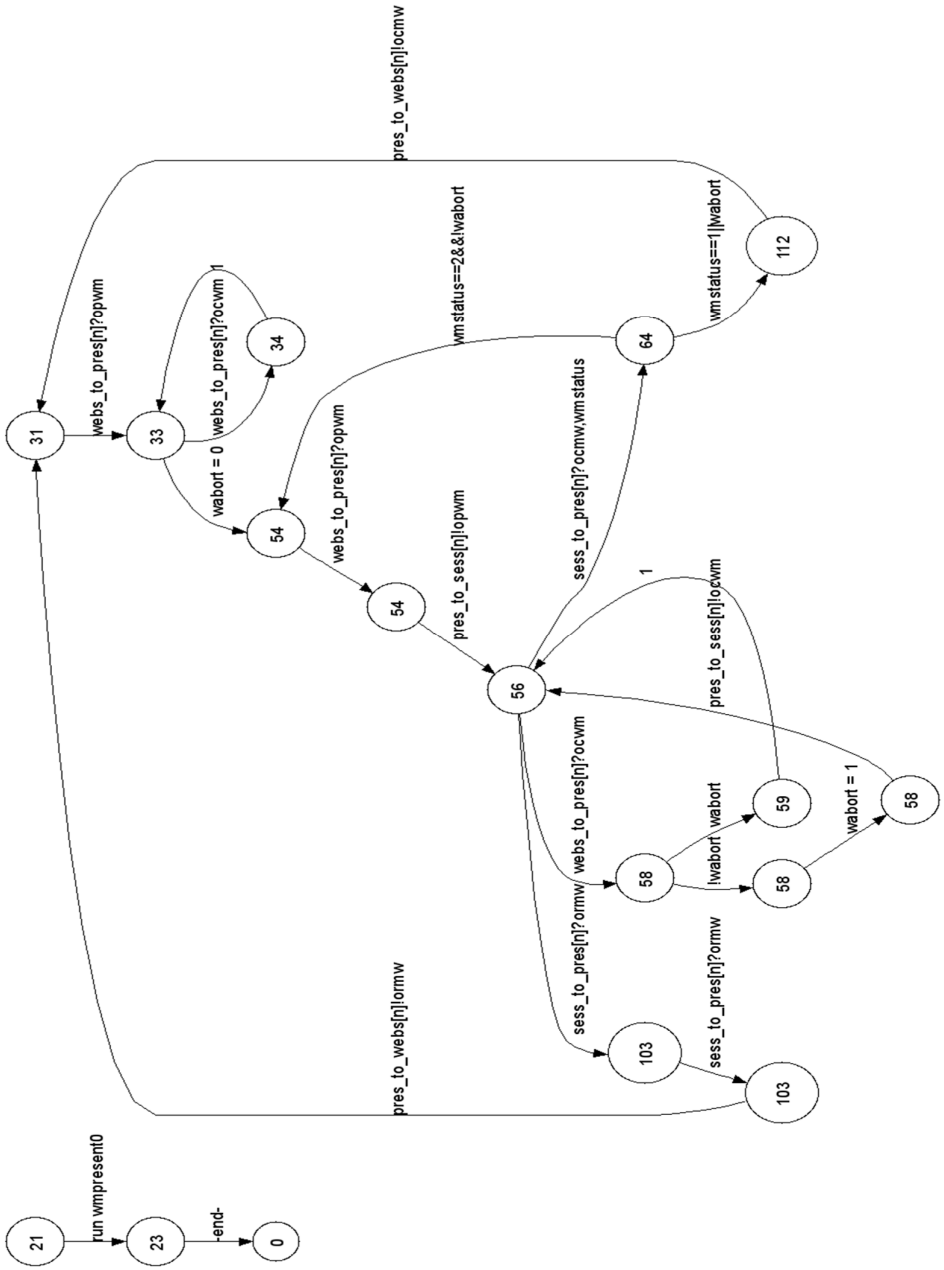

Fig. 9. State chart diagram for wmpresent process in WMprotocol.

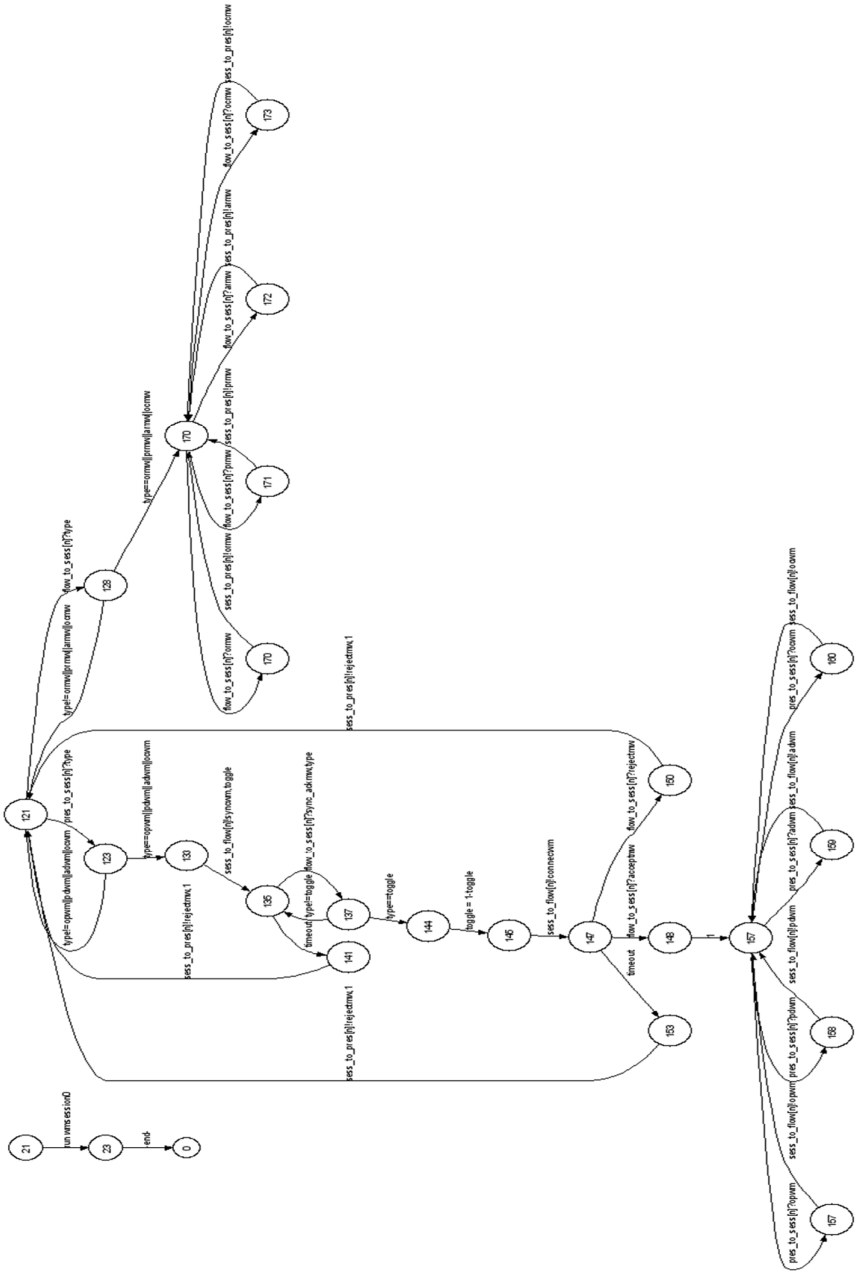

Fig. 10. State chart diagram for wmsession process in WMprotocol.

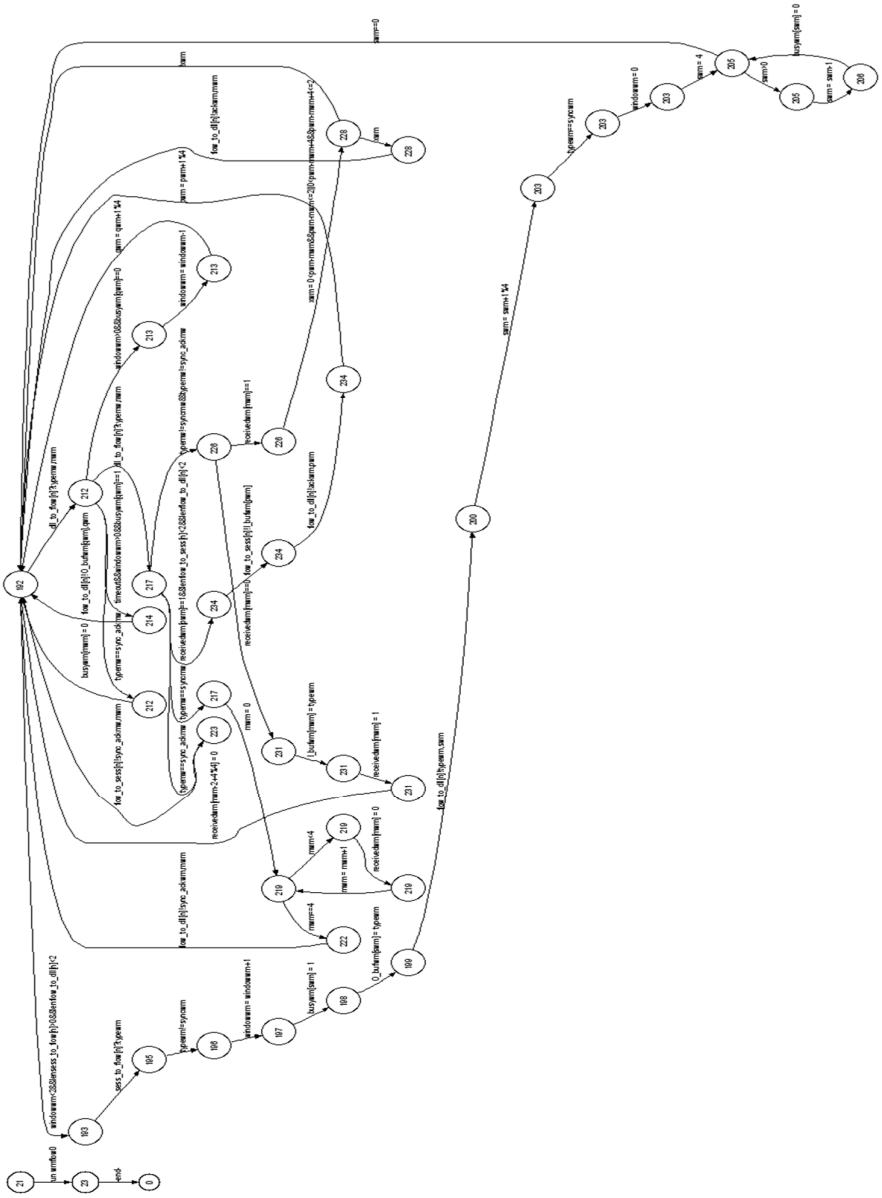

Fig. 11. State chart diagram for wmflow process in WMprotocol.

4. Properties related to WM protocol

For the WM protocol, model specifications are represented in PROMELA, and SPIN model checker helps identify the unreachable code and deadlocks in the given model. The properties of wm protocol are represented in LTL formulas in Table 1.

Tab. 1. LTL Formulas of WM protocol Model

1	[]<u>(wabort||!wabort)</u>
2	[]<u>(wabort&&!wabort)</u>
3	[](opwm-><>ormw)
4	[](pdwm-><>prmw)
5	[](adwm-><>armw)

5. Verification Results

The LTL formulas shown in Table 1 are verified using spin model checker, and the corresponding results are reported.

[](wabort||!wabort)

```
(Spin Version 6.0.0 -- 5 December 2010)
     + Partial Order Reduction
Full statespace search for:
     never claim          +
     assertion violations  + (if within scope of claim)
     cycle checks           - (disabled by -DSAFETY)
     invalid end states  - (disabled by never claim)
State-vector 168 byte, depth reached 5, --- errors: 0 ---
     2 states, stored
     1 states, matched
     3 transitions (= stored+matched)
     2 atomic steps
hash conflicts:          0 (resolved)
     2.195  memory usage (Mbyte)
unreached in init
     (0 of 5 states)
unreached in proctype wmpresent
     (28 of 120 states)
unreached in proctype wmsession
     (22 of 72 states)
unreached in proctype wmflow
     (24 of 71 states)
unreached in claim p1
     _spin_nvr.tmp:8, state 8, "-end-"
     (1 of 8 states)
pan: elapsed time 0.015 seconds
pan: rate 133.33333 states/second
```

Fig. 12. Verification result for LTL formula 1.

[](wabort&&!wabort)

```
(Spin Version 6.0.0 -- 5 December 2010)
Warning: Search not completed
     + Partial Order Reduction
Full statespace search for:
     never claim          +
     assertion violations  + (if within scope of claim)
     cycle checks           - (disabled by -DSAFETY)
     invalid end states  - (disabled by never claim)
State-vector 168 byte, depth reached 5, *** errors: 1 ***
     2 states, stored
     0 states, matched
     2 transitions (= stored+matched)
     2 atomic steps
hash conflicts:          0 (resolved)
     2.195  memory usage (Mbyte)
pan: elapsed time 0 seconds
```

Fig. 13. Verification result for LTL formula 2.

[](opwm-><>ormw)

```
(Spin Version 6.0.0 -- 5 December 2010)
     + Partial Order Reduction
Full statespace search for:
     never claim          +
     assertion violations  + (if within scope of claim)
     cycle checks         - (disabled by -DSAFETY)
     invalid end states  - (disabled by never claim)
State-vector 168 byte, depth reached 5, --- errors: 0 ---
     2 states, stored
     1 states, matched
     3 transitions (= stored+matched)
     2 atomic steps
hash conflicts:          0 (resolved)
     2.195  memory usage (Mbyte)
unreached in init
     (0 of 5 states)
unreached in proctype wmpresent
     (28 of 120 states)
unreached in proctype wmsession
     (22 of 72 states)
unreached in proctype wmflow
     (24 of 71 states)
unreached in claim p1
     _spin_nvr.tmp:8, state 9, "(!(ormw))"
     _spin_nvr.tmp:10, state 11, "-end-"
     (2 of 11 states)
pan: elapsed time 0 seconds
```

Fig. 14. Verification result for LTL formula 3.

[](pdwm-><>prmw)

```
(Spin Version 6.0.0 -- 5 December 2010)
        + Partial Order Reduction
Full statespace search for:
        never claim             +
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)
State-vector 168 byte, depth reached 5, --- errors: 0 ---
        2 states, stored
        1 states, matched
        3 transitions (= stored+matched)
        2 atomic steps
hash conflicts:         0 (resolved)
        2.195  memory usage (Mbyte)
unreached in init
        (0 of 5 states)
unreached in proctype wmpresent
        (28 of 120 states)
unreached in proctype wmsession
        (22 of 72 states)
unreached in proctype wmflow
        (24 of 71 states)
unreached in claim p1
        _spin_nvr.tmp:8, state 9, "(!(prmw))"
        _spin_nvr.tmp:10, state 11, "-end-"
        (2 of 11 states)
pan: elapsed time 0 seconds
```

Fig. 15. Verification result for LTL formula 4.

[](adwm-><>armw)

```
(Spin Version 6.0.0 -- 5 December 2010)
        + Partial Order Reduction
Full statespace search for:
        never claim             +
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)
State-vector 168 byte, depth reached 5, --- errors: 0 ---
        2 states, stored
        1 states, matched
        3 transitions (= stored+matched)
        2 atomic steps
hash conflicts:         0 (resolved)
        2.195  memory usage (Mbyte)
unreached in init
        (0 of 5 states)
unreached in proctype wmpresent
        (28 of 120 states)
unreached in proctype wmsession
        (22 of 72 states)
unreached in proctype wmflow
        (24 of 71 states)
unreached in claim p1
        _spin_nvr.tmp:8, state 9, "(!(armw))"
        _spin_nvr.tmp:10, state 11, "-end-"
        (2 of 11 states)
pan: elapsed time 0 seconds
```

Fig. 16. Verification result for LTL formula 4.

6. Conclusion

The WM protocol related to B2C e-commerce was designed by considering service specification, environment, protocol vocabulary, and message format and procedure rules. The procedure rules related to the WM protocol were specified by PROMELA language. The state chart diagrams of WM protocol processes were given. The properties of WM protocol were specified in linear temporal logic. The verification of properties was performed by SPIN model checker, and the corresponding results were reported.

7. Acknowledgements

The author acknowledges University Grants Commission (UGC) for providing support and resources needed to carry out this study by sanctioning minor research project.

References

[1] Gerard J. Holzmann. Design and Validation of Computer Protocols. Prentice Hall, (1991).

[2] Holzmann, G. J., "The Model Checker SPIN," IEEE Transactions on Software Engineering, Vol. 23, No. 5, (1997).

[3] Spin Manual, Online: http://spinroot.com/spin/Man/Manual.html.

[4] Mordechai Ben-Ari " Principles of the Spin Model Checker" springer, (2007).

[5] Iqra Obaid, Syed Asad Raza Kazmi, Awais Qasim "Modeling and Verification of Payment System in E-Banking "(IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 8, No. 8, (2017).

[6] Huiling Shi , Wenke Ma, Meihong Yang and Xinchang Zhang , "A Case Study of Model Checking Retail Banking System with SPIN"JOURNAL OF COMPUTERS, VOL. 7, NO. 10, OCTOBER (2012).

[7] Syed MS Islam, Mohammed H Sqalli, and Sohel Khan. Modeling and formal verification of dhcp using spin. IJCSA, Vol. 3, No. 2, (2006), pp. 145-159,

[8] Wei Zhang. "Model checking and verification of the internet payment system with spin" Journal of Software, (2012), pp. 235- 257,

[9] Haiping Xu and Yi-Tsung Cheng. Model checking bidding behaviors in internet concurrent auctions. International Journal of Computer Systems Science & Engineering, Vol. 22, No. 4, (2007), pp. 179 -191,

[10] Xu, S., Matthews, M.M., Huang, C.T., "Modeling and Analysis of IEEE 802.16 PKM Protocols using CasperFDR," IEEE ISWCS (2008).

[11] Krishnam Raju, K.V., Valli kumari, V., Sandeep varma, N., Raju, KVSVN., "Formal Verification of IEEE802.16m PKMv3 Protocol Using CasperFDR," ICT, Springer LNCS-CCIS, Vol. 101, No. 3, (2010), pp. 590-595.

[12] Kim, I.G., Kim, H.S., Lee, J.Y., Choi, J.Y., "Analysis and Modification of ASK Mobile Security Protocol," WMCS, IEEE Xplore (2005).

[13] Ramasamy, H.V., Cukier, M., Sanders, W.H., "Formal Verification of an Intrusion-Tolerant Group Membership Protocol," IEICE Transactions (2003).

[14] Krishnam Raju, K.V., Valli Kumari, V., Raju, KVSVN., "Modeling and Analysis of IEEE802.11i WPA-PSK Authentication Protocol," ICNCS, IEEE Xplore, (2011), pp. 72-76.

[15] Krishnam Raju, K.V., Valli Kumari, V., "Formal Verification of IEEE802.11i WPA-GPG Authentication Protocol," AIM, Springer LNCS-CCIS, Vol. 147, No. 2, (2011), pp. 267-272.

[16] Kazhamiakin, R., Pistore, M. and Roveri, M., "Formal verification of requirements using SPIN: a case study on Web services", In Proceedings of the Second International Conference on Software Engineering and Formal Methods, SEFM 2004, 28-30. (2004), pp. 406 - 415.

[17] Ribeiro, O.R., Fernandes, J.M. and Pinto, L.F., "Model checking embedded systems with PROMELA", In Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, ECBS '05. 4-7 (2005), pp. 378 - 385.

[18] Li Yongjian, and Xue Rui, "Using SPIN to model cryptographic protocols", In Proceedings of International Conference on Information Technology: Coding and Computing,. ITCC 2004, Vol. 2, (2004), pp. 741- 745.

[19] Gluck, P.R. and Holzmann, G.J, "Using SPIN model checking for flight software verification", In Proceedings of IEE Aerospace Conference, Vol. 1, (2002), pp.1-105 - 1- 113.

[20] Havelund, K., Lowry, M. and Penix, J., "Formal analysis of a space-craft controller using SPIN", IEEE Transactions on Software Engineering, Vol. 27, No. 8, (2001), pp. 749 - 765.

Follow This Article at The Following Site:

Krishnamraju K. Design and verification of WM protocol for Electronic Commerce Transactions. IJIEPR. 2020; 31 (2) :323-338
URL: http://ijiepr.iust.ac.ir/article-1-1060-en.html