

# Formal Model and Analysis of Sliding Window Protocol Based on NuSMV

Yefei Zhao

Department of Computer Science, East China Normal University, Shanghai, China  
Email: derekzhaoecnu@gmail.com

YANG Zong-yuan, Jinkui Xie and Qiang Liu

Department of Computer Science, East China Normal University, Shanghai, China  
Email: zyuan@cs.ecnu.edu.cn, jkxie@cs.ecnu.edu.cn, lqiangecnu@gmail.com

**Abstract**—System states space based on Kripke structure can be exhausted by model checking, thus system key property described by temporal logic can be automatically verified. Presently model checking has been widely used in hardware validation and network protocol analysis. Sliding window protocol is a classical receive-send protocol, which is used in TCP/IP protocol group. In this paper, we propose the respective formal model of sliding window protocol under three conditions, as well as Kripke structure semantics of the protocol. The key properties of system such as data integrity, liveness and information consistency are automatically validated. Finally experiment, table and analysis are given out. The method we proposed to analysis specific bit sliding window protocol can be extended to analysis arbitrary bit sliding window protocol.

**Index Terms**—Sliding window protocol, Model Checking, Protocol analysis, NuSMV, CTL

## I. INTRODUCTION

Model checking ([3], [4]) is a formal validation technology comprised of temporal logic and automata theory. System states spaces are described by Kripke structure, key system property is described by temporal logic (Computation Tree Logic or Linear Temporal Logic), thus system states space can be exhausted to verify whether the key property holds or not. Presently model checking has been widely used in hardware validation ([14]), network protocol analysis ([8], [9]) and critical security system validation.

It is an effective method to analysis network protocol by model checking. Some research works have been done such as: In [10], wireless authentication protocol is validated. In [11], security protocol is modeled and validated.

In ABP protocol, page number that sender sends to receiver and acknowledgement number that receiver sends to sender are either 0 or 1, which can be easily described in model checker. However, in sliding window

protocol, the page number, acknowledgement number and size of sliding window can be any integer, which can't be processed in model checking. To solve the problem, we propose the abstraction model of sliding window protocol and take into account system concurrency.

The rest of this paper is structured as follows: Section 2 describes the theory of NuSMV; Section 3 describes the formal model of sliding window protocol; in section 4, the formal model in case of intruder attacking is represented; in section 5, the key property of system such as data integrity, liveness and information consistency are automatically verified in NuSMV under three conditions. Section 6 describes conclusion and future work.

## II. KRIPKE STRUCTURE AND NuSMV

Kripke structure is a 4-tuple  $M_k = (Q_k, I_k, \Delta_k, L_k)$  where:  $Q_k$  is a finite set of states,  $I_k \subseteq Q_k$  is the set of initial states,  $\Delta_k \subseteq Q_k \times Q_k$  is a transition relation which represents a state and its successor states,  $L_k : Q_k \rightarrow 2^{AP}$  is a function which returns the set of atomic propositions that hold true in a state.

Kripke structure is used to represent the static topology and dynamic behavior of a system. Each state relates with a set of atomic propositions.

NuSMV ([5], [6], [7]) is an automatic model checker to verify system property described by temporal logic. In NuSMV, the value transition of variable indirectly represents the transitions of states. NuSMV input language is the abstract notations of Kripke structure.

The basic system diagram of NuSMV is illustrated as Fig. 1.

There are four basic steps in model checking:

1. We represent formal model of system with NuSMV input language, which includes system states and transitions between system states.

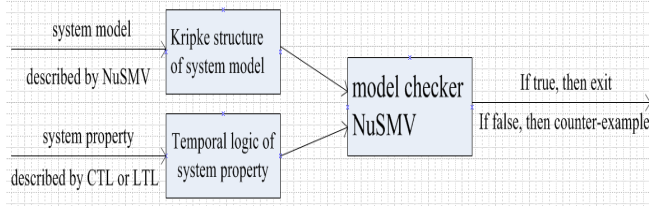


Figure 1. Basic theory diagram of NuSMV

2. Key system property to be validated is described by temporal logic such as CTL or LTL.
3. Inside model checker, system states, transitions between system states and system properties to be validated are all represented by BDD structure.
4. Key system properties are automatically reasoned about and verified by model checker.

In NuSMV, system states and transitions between system states are described with Kripke structure, key system properties are described with temporal logic such as CTL or LTL.

III. FORMAL MODEL OF SLIDING WINDOW PROTOCOL

Sliding window protocol comprises of three parts: sender, receiver and channel, which works as follows:

1. Sender  $\xrightarrow{message}$  Receiver { message is an arbitrary page in sending window }
2. Receiver  $\xrightarrow{Ack}$  Sender { after receiver received data page, if the number of data page is in receiving window, then Receiver sent the acknowledgement of this page to sender }
3. If all data pages in sending window received acknowledgements, then sending window slides forward.
4. If receiver received all data pages in receiving window, then receiving window slides forward.

Steps (1) and (2) execute asynchronously. Sender can send arbitrary data page in sending window, Receiver can receive any data page in receiving window, then return acknowledgement of this page.

For more detailed information about sliding window protocol, please refer to [2].

There are  $(2^n - 1)$  states in n-variable system in NuSMV. In this paper, a classical 2-bit sliding window protocol is illustrated, where the size of sliding window is 4, the size of data channel is 3 bits and the size of the size of acknowledgement channel is 4 bits. The system diagram of sliding window protocol which includes sender, receiver and channel is shown as Fig. 2.

Sender takes 4-bit acknowledgement channel as input, as well 3-bit data channel as output to send data page and page number. Sender module is defined in NuSMV as follows:

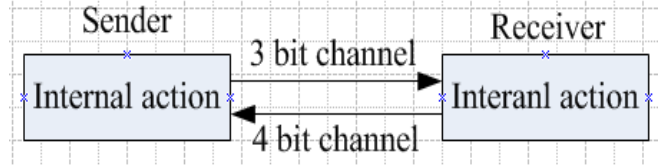


Figure 2. System diagram of sliding window protocol

s : process sender(channel4.output1, channel4.output2, channel4.output3, channel4.output4);

Fig. 3 represents the finite states automata model of sender module. S is the abbreviation of sender,  $s_0$  represents the initial state, “?” represents receiving action, “!” represents sending action,  $m_i$  represents the ith data page,  $ack_j$  represents the acknowledgement of the jth data page, “receive all 4 ack” represents that receiver received all the four acknowledgements in sending window, “slide window” represents sending/receiving window slides forward, e represents null action.

Sender randomly selects a data page in sending window and sends it to Receiver, retry in case of timeout. After Sender receives an acknowledgement of some data page, it will check whether all the four acknowledgements arrived, if true, then sending window slides forward. Sending window and receiving window separately slide circularly. The actions that Sender sends data page and receives acknowledgement are asynchronously.

After Sender receives acknowledgement, it will check whether all the four acknowledgements have arrived, if true, then sending window slides forward. The according NuSMV input language is as follows:

```
next(bit0_send) :=
case
  (bit0_ack = bit0_send) & (bit1_ack = bit1_send)
  & (bit2_ack = bit2_send) & (bit3_ack=bit3_send) :
  !(bit0_send);
  1 : bit0_send;
esac;
```

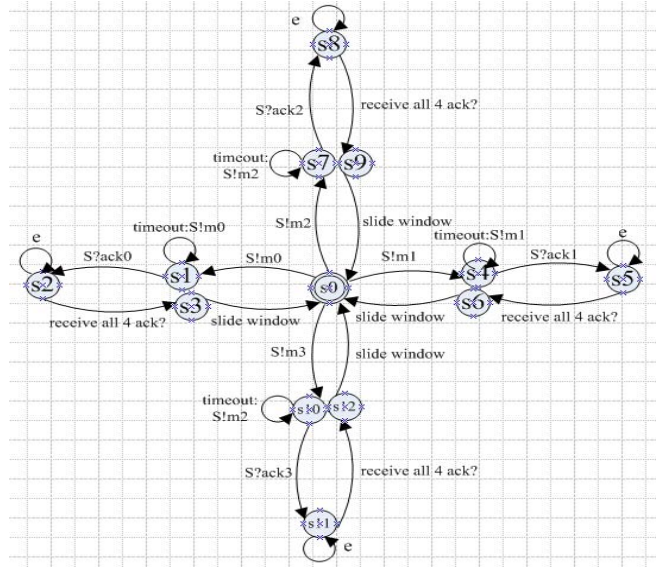


Figure 3. Finite states automata model of Sender

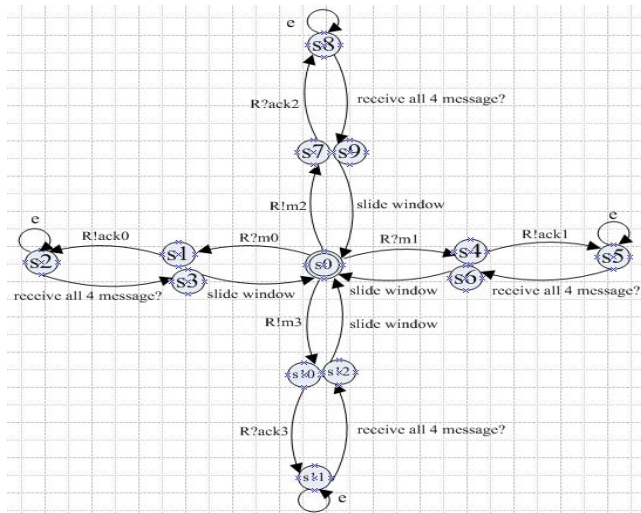


Figure 4. Finite states automata model of Receiver

In NuSMV input language, the keyword “FAIRNESS” is used to guarantee path fairness such that none path will be never scheduled by model checker, which makes it possible to formally model the sliding window protocol under the condition that data page maybe lost in channel. The according NuSMV input language as follows:

```
next(output1) := case
    forget : output1;
    1 : input1;
esac;
```

```
.....
FAIRNESS input1 & !forget
FAIRNESS !input1 & !forget
```

Similarly, the finite states automata model of Receiver in sliding window protocol is represented in Fig. 4.

#### IV. FORMAL MODEL IN CASE OF INTRUDER

In this section, we will validate the security property of sliding window protocol in case of Intruder existing, which maybe capture, modify and resend the data page and acknowledgement that it received. Intruder comprises of three parts: 3-bit channel, 4-bit channel and internal action. Intruder module can interact with Sender module and Receiver module through 3-bit channel and 4-bit channel. The internal actions of Intruder module includes: replacing Sender, replacing Receiver and monitoring channel.

The system diagram of sliding window protocol which includes Sender, Receiver and Intruder is shown in Fig. 5.

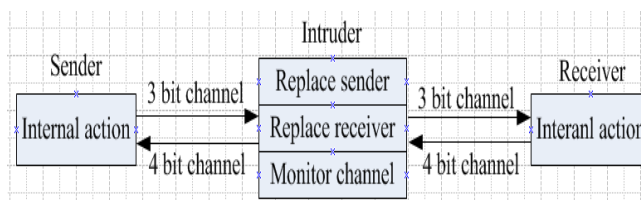


Figure 5. System diagram of sliding window protocol with Intruder

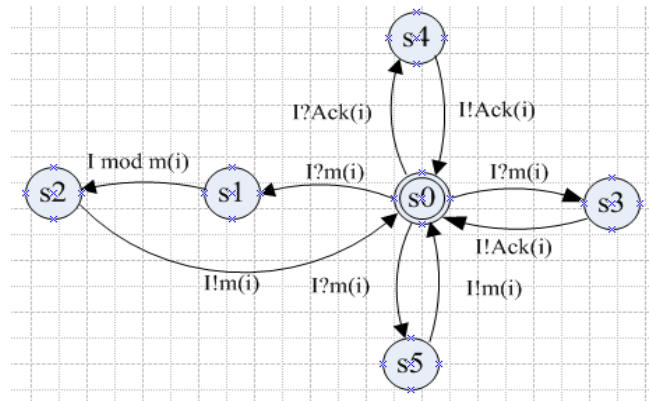


Figure 6. Finite states automata model of Intruder

Intruder comprises of 3-bit channel and 4-bit channel. In case Intruder tries to replace Sender, Intruder will modify and transmit data page after receiving it, thus Receiver receives the data pages that have been modified by Intruder. The according NuSMV input language is as follows:

```
next(output1) := case
    forget : input1;
    1 : !(input1);
esac;
```

The activities that Intruder replaces Receiver and monitors channel are similarly as above. The finite states automata model of Intruder is shown in Fig. 6.

In Fig. 6, “I” is the abbreviation of Intruder, Ack(i) represents the acknowledgement of the ith data page, m(j) represents the jth data page, “mod” is the abbreviation of modify.

#### V. AUTOMATIC VERIFICATION AND ANALYSIS

##### A. Experiment environment

Software and hardware environment of the experiment are: Operating system is RedHat 7.2, Pentium 2.4G, 1G memory, model checker NuSMV 2.4.2.

There are many system properties in sliding window protocol. We focus on some important properties described by CTL formula under three conditions: idea channel, message maybe lost in channel and under attack.

##### B. Syntax and semantic of CTL

CTL (Computation Tree Logic) is a kind of branch time logic. The BNF diagram of CTL syntax is as follows:

$$\Phi ::= \perp \mid T \mid p \mid (\neg\Phi) \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi) \mid (\Phi \rightarrow \Phi) \mid AX\Phi \mid EX\Phi \mid AF\Phi \mid EF\Phi \mid EG\Phi \mid A[\Phi U\Phi] \mid E[\Phi U\Phi]$$

Every CTL temporal connector is a pair of symbols.

The first symbol of the pair is either “A” or “E” where “A” represents “all path” and “E” represents “exist a path”. “A” and “E” are path operator.

The second symbol of the pair is possibly “X”, “F”, “G” and “U”. “X” represents “next state”, “F” represents “some future state”, “G” represents “all cases in future”,

“U” represents “until”. “X”, “F”, “G” and “U” are temporal operator.

A path operator must accompany with a temporal operator, and vice versa.

$M=(S, \rightarrow, L)$  is a Kripke structure model.  $S$  is a set of system states.  $L$  is the set of system labels. “ $\rightarrow$ ” is a function defined as:  $S \rightarrow L$ . Function “ $L$ ” relates a state to a set of labels.

For more detailed information about CTL, please refer to [1], [13].

C. Ideal channel

**Definition 1 Information consistency:** If Sender sent a data page and received the acknowledgement of this page, then the data page that Receiver received is consistent with what Sender sent.

In case of ideal channel, information consistency is described by CTL formula as follows:

$AG ( s.st = sending \rightarrow AF s.st = sent ) \ \& \ AG ( r.st = receiving \rightarrow AF r.st = received )$

**Definition 2 Liveness:** If Sender is in the state of “sending”, then Sender will arrive at the state of “sent”. Similarly, if Receiver is in the state of “receiving”, then Receiver will arrive at the state of “received”.

Liveness is described by CTL formula as follows:

$AG ( s.st = sending \rightarrow AF s.st = sent ) \ \& \ AG ( r.st = receiving \rightarrow AF r.st = received )$

**Definition 3 Data integrity:** If the state of Receiver transforms from “receiving” to “received”, then it can imply that the state of Sender transforms from “sending” to “sent”.

The NuSMV input language of system main module for sliding window protocol is as follows:

```
MODULE main
VAR
s : process sender(channel_4_intruder.output1,
channel_4_intruder.output2, channel_4_intruder.output3,
channel_4_intruder.output4);
r : process receiver(channel_3_intruder.output1,
channel_3_intruder.output2, channel_3_intruder.output3);
channel_4_intruder : process
four_bit_intruder(r.bit0_return, r.bit1_return,
r.bit2_return, r.bit3_return);
channel_3_intruder : process
three_bit_intruder(s.message1, s.bit0_page, s.bit1_page);
```

From above source code, we can see that module main comprises of four parts: Sender, Receiver, 3-bit channel and 4-bit channel.

Below is a part of definitions and activities of module Sender in NuSMV:

```
<root@localhost /home/derek/NuSMV/nusmv/examples/2bit-abp># nusmv 2bit_abp_2channel_model.smv
*** This is NuSMV 2.4.2 (compiled on Tue Jul 29 20:53:34 UTC 2008)
*** For more information on NuSMV see <http://nusmv.first.itc.it>
*** or email to <nusmv-users@irst.itc.it>.
*** Please report bugs to <nusmv@irst.itc.it>.

-- specification AG (s.st = sending -> AF s.st = sent) & AG (r.st = receiving -> AF r.st = received) is true
-- specification AG (r.st = received -> AF s.st = sent) is true
-- specification AG (s.st = sent -> s.message1 = r.receive_message) is true
<root@localhost /home/derek/NuSMV/nusmv/examples/2bit-abp>#
```

Figure 7. Automatic verification under ideal channel

```
MODULE sender(bit0_ack, bit1_ack, bit2_ack, bit3_ack)
VAR
st : { sending, sent };
bit0_page : boolean;
bit1_page : boolean;
message1 : boolean;
bit0_message : boolean;
bit1_message : boolean;
bit2_message : boolean;
bit3_message : boolean;
bit0_send : boolean;
bit1_send : boolean;
bit2_send : boolean;
bit3_send : boolean;
ASSIGN
init(st) := sending;
next(st) := case
st = sent : sending;
(bit0_ack = bit0_send) & (bit1_ack =
bit1_send) & (bit2_ack = bit2_send) & (bit3_ack =
bit3_send) & !(st = sent) : sent;
1 : sending; esac;
```

..... Under ideal channel, NuSMV model checker automatically verified these properties as Fig. 7.

We can extract the automatic verification result in Fig. 7 as shown in Table 1.

TABLE I. RESULT UNDER IDEAL CHANNEL

Case	CTL specification	Result
Ideal channel	$AG ( s.st = sent \rightarrow s.message1 = r.receive\_message )$	true
	$AG ( s.st = sending \rightarrow AF s.st = sent ) \ \& \ AG ( r.st = receiving \rightarrow AF r.st = received )$	true
	$AG ( r.st = received \rightarrow AF s.st = sent )$	true

From Table 1 we can see that the experiment results are all true, which means that information consistency, liveness and data integrality hold true in sliding window protocol.

*D. Message maybe lost in channel*

The key properties to be verified is same as “Ideal channel”, thus the CTL formulas is also the same.

The NuSMV input language of 3-bit channel and 4-bit channel in sliding window protocol under the condition that message maybe lost is defined as follows:

```

MODULE three_bit_channel(input1, input2, input3)
  VAR
    output1 : boolean;
    output2 : boolean;
    output3 : boolean;
  ASSIGN
    next(output1) := case
      1 : !(input1);  esac;
    next(output2) := case
      1 : input2;    esac;
    next(output3) := case
      1 : input3;    esac;
  FAIRNESS running
MODULE four_bit_intruder(input1, input2, input3, input4)
  VAR
    output1 : boolean;
    output2 : boolean;
    output3 : boolean;
    output4 : boolean;
  ASSIGN
    next(output1) := case
      1 : input1;  esac;
    next(output2) := case
      1 : input2;  esac;
    next(output3) := case
      1 : input3;  esac;
    next(output4) := case
      1 : input4;  esac;
  FAIRNESS running
  
```

Under the condition that message maybe lost in channel, NuSMV model checker automatically verified the properties as Fig. 8.

```

<root@localhost /home/derek/NuSMV/nusmv/examples/2bit-abp># nusmv 2bit_abp_2chan
nel_lost.smv
*** This is NuSMV 2.4.2 (compiled on Tue Jul 29 20:53:34 UTC 2008)
*** For more information on NuSMV see <http://nusmv.irst.itc.it>
*** or email to <nusmv-users@irst.itc.it>.
*** Please report bugs to <nusmv@irst.itc.it>.

-- specification AG (s.st = sent -> s.message1 = r.receive_message) is true
-- specification (AG (s.st = sending -> AF s.st = sent) & AG (r.st = receiving -
> AF r.st = received)) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  s.st = sending
  s.bit0_page = 0
  s.bit1_page = 0
  s.message1 = 0
  
```

Figure 8. Automatic verification under message maybe lost

We extract the automatic verification result in Fig. 8 as shown in Table 2.

From Tabel 2 we can see that: Under the condition that message maybe lost, information consistency holds true while liveness and data integrality are false. Next we will discuss why properties liveness and data integrality are destroyed.

From the counter-example NuSMV provided, we see that receiving window slides forward after Receiver received all the data pages, however, the acknowledgement lost in channel when Receiver sent it to Sender. After that, Sender kept retrying because of timeout. Next we will discuss it under two conditions:

1. Sending window has no overlap on receiving window: Sender sends data pages again because of timeout, Receiver received the data page which isn't in receiving window, then discarded it. After this, Sender kept in the state of "timeout", while Receiver kept discarding the received data page, which makes system deadlock.
2. Sending window has overlap on receiving window: Receiver received the origin data page, regarded it as a new data page in receiving window and accepted it, which makes sliding window protocol fail. Because: After receiving window of Receiver slides forward, new receiving window has overlap on old receiving window. To guarantee there are not any overlaps, the maximum size shouldn't exceed half of page number.

TABLE II.  
RESULT UNDER MESSAGE MAYBE LOST IN CHANNEL

Case	CTL specification	Result
Message maybe lost	AG ( s.st = sent -> s.message1 = r.receive_message)	true
	AG ( s.st = sending -> AF s.st = sent ) & AG ( r.st = receiving -> AF r.st = received )	false
	AG ( r.st = received -> AF s.st = sent )	false

### E. Under attack

The internal activities of Intruder comprise of replacing Sender, replacing Receiver and monitoring channel. The key properties to be verified are different from what are defined in section 5.1, so we propose new definition of these properties.

Intruder maybe modifies the content of data page, so the property of information consistency is described with CTL formula as follows:

AG ( s.st = sent -> s.message1 = r.receive\_message )

Intruder maybe replaces Sender or Receiver, so the property of liveness should be verified.

**Definition 4 Liveliness:** The state of Sender transforms from “sending” to “sent”, the state of Intruder transforms from “receiving” to “received” while the state of Receiver still keeps “receiving”.

The property of liveness represents whether Intruder replaces Sender or Receiver or not, which is described by CTL formula as follows:

AG ( s.st = sent -> intruder.st = received & r.st = receiving )

The NuSMV input language of Intruder in sliding window protocol under attack is defined as follows:

```

MODULE three_bit_intruder(input1, input2, input3)
  VAR
    output1 : boolean;
    output2 : boolean;
    output3 : boolean;
  ASSIGN
    next(output1) := case
      1 : !(input1);  esac;
    next(output2) := case
      1 : input2;    esac;
    next(output3) := case
      1 : input3;    esac;
FAIRNESS running
MODULE four_bit_intruder(input1, input2, input3, input4)
  VAR
    output1 : boolean;
    output2 : boolean;
    output3 : boolean;
    output4 : boolean;
  ASSIGN
    next(output1) := case
      1 : input1;  esac;

```

```

<root@localhost /home/derek/NuSMV/nusmv/examples/2bit-abp># nusmv 2bit_abp_repla
ce_sender.smv
*** This is NuSMV 2.4.2 (compiled on Tue Jul 29 20:53:34 UTC 2008)
*** For more information on NuSMV see <http://nusmv.irst.itc.it>
*** or email to <nusmv-users@irst.itc.it>.
*** Please report bugs to <nusmv@irst.itc.it>.

-- specification AG (s.st = sent -> s.message1 = r.receive_message) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  s.st = sending
  s.bit0_page = 0
  s.bit1_page = 0
  s.message1 = 0
  s.bit0_message = 0
  s.bit1_message = 0
  s.bit2_message = 0
  s.bit3_message = 0
  s.bit0_send = 1
  s.bit1_send = 1

```

Figure 9. Automatic verification under attack

```

next(output2) := case
  1 : input2;  esac;
next(output3) := case
  1 : input3;  esac;
next(output4) := case
  1 : input4;  esac;

```

FAIRNESS running

We can extract the automatic verification result in Fig. 9 as shown in Table 3.

The result for property of information consistency is false. From the counter-example we can see: Intruder successfully captured and modified the data page that Sender sent, then transformed it to Receiver, Receiver received the wrong data page. Intruder successfully replaced Sender.

The result for property of liveness is true. We can conclude that Intruder successfully captured the data page Sender sent and sent the acknowledgement of this page directly to Receiver, which made Receiver keep in the state of “receiving”. Intruder successfully replaces Receiver.

Protocol designer can introduce cryptogram and authentication to resolve these problems.

## VI. CONCLUSION AND FUTURE WORK

From the experiment in section 5, we can conclude:

1. In case of ideal channel, the key properties hold true in sliding window protocol.

TABLE III.  
RESULT UNDER ATTACK

Case	CTL specification	Result
Under attack	AG ( s.st = sent -> s.message1 = r.receive_message )	false
	AG ( s.st = sent -> intruder.st = received & r.st = receiving )	true

2. In case of message maybe lost in channel, the property of information consistency holds true, while the properties of liveness and data integrality are false.
3. In case of under attack, the properties of information consistency, liveness and data integrality are all false.
4. Protocol designer can find weakness of protocol according to counter-example NuSMV provided and propose mechanism to solve these problems.
5. Model checking is an effective method to do hardware validation and network protocol analysis.

In this paper, a classical 2-bit sliding window protocol is illustrated to do network protocol analysis. The basic method can be extended to n-bit sliding window protocol.

The sliding window protocol we analysis in this paper is classical. Nevertheless, there are some other extensions of sliding window protocol, for example n-page retreat technology, selection transformation and so on. In the future, we will do some advance research about these protocols.

Electronic commerce has been widely used in Internet. More and more researches ([12], [13]) have been done to guarantee the properties of security, atomic and authentication of electronic commerce protocol. We will do some researches about electronic commerce protocol analysis.

Model checking have been widely used in function verification, however it is a new start to do non-function validation and performance analysis with model checking, probabilistic model checking is proposed to resolve these problems ([15], [16]). We will do some research about probabilistic model checking in the future.

#### ACKNOWLEDGMENT

The work was supported by the National Natural Science Foundation of China under Grant No.60703004; the National Research Fund for the Doctoral Program of Higher Education of China under Grant No. 20060269002; PhD Program Scholarship Fund of ECNU 2007.

#### REFERENCES

- [1] Michael Huth, Mark Ryan. Logic in Computer Science. Cambridge Press, 2004
- [2] Tanenbaum A S. Computer Networks. Prentice Hall PTR, 1996, 4(3): 279-297.
- [3] Clarke E , Grumberg O , Peled D. Model checking. MIT Press , 2000.
- [4] McMillan KL. Symbolic model checking [D].PhD thesis.University of Carnegie Mellon, Pittsburgh, USA, 1992
- [5] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV : a new symbolic model checker. International Journal on Software Tools for Technology Transfer (STTT), 2(4), March 2000. 359
- [6] Cimatti A. , Clarke E. M. , Giunchiglia F. , Giunchiglia F. , Pistore M. ,Roveri M. , Sebastiani R. , Tacchella A. . NuSMV Version 2 : An OpenSource tool for symbolic model checking. In : Proceedings of the International Conference on Computer-Aided Verification ( CAV 2002 ) , Copenhagen , Denmark ,2002 , 359 ~ 364
- [7] McMillan K. SMV[Z]. <http://www-cad.eecs.berkeley.edu/~kenmcmil>, 2004-06.
- [8] Kim Il- Gon, Choi Jin-Young. Formal verification of PAP and EAPMD5 protocols in wireless networks: FDR model checking[C].In: Proceedings of the 18th International Conference on Advanced Information Networking and Applications, Fukuoka, 2004: 29~31
- [9] Shim Kyungah. Cryptanalysis of mutual authentication and key exchange for low power wireless communications[J].IEEE Communications Letters, 2003; 7( 5) : 248~250
- [10] Marrero W, Clarke E, Jha S. A Model Checker for Authentication Protocols[C]. DIMACS Workshop on Design and Formal Verification of Security Protocols, 1997:134-141.
- [11] Will Marrero, Edmund Clarke, Somesh Jha. Model checking for security protocols. Carnegie Mellon University: Technical Report CM U—SCS~97—139, 1997
- [12] Mitechell J, Mitechell M, Stern U. Automated Analysis of Cryptographic Protocols Using Mur[C]. Proc. of the IEEE Symposium on Security and Privacy, 1997: 141-151.
- [13] Kesten Y, Pnueli A. A compositional approach to ctl verification. Theoretical Computer Science , 2005 , 331 (2 ,3) : 397 ~ 428
- [14] Burch J. R, Clarke E. M , Long D. E. , MacMillan K. L, Dill D. L. Symbolic model checking for sequential circuit verification. IEEE Transactions on ComputerAided Design of Integrated Circuits and Systems , 1994 , 13 (4) : 401~424
- [15] R. Bakhshi, F. Bonnet, W. Fokink, and B. Haverkort. Formal analysis techniques for gossiping protocols. ACM SIGOPS Operating Systems Review, 41(5):28 - 36, 2007.
- [16] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In Proc. TACAS'06, volume 3920 of LNCS, pages 441 - 444. Springer, 2006.
- [17] Yefei Zhao, YANG Zong-yuan, Jinkui Xie. Formal semantics of UML state diagram and automatic verification Based on Kripke structure. 22nd IEEE

Yefei Zhao was born in Jinlin city, China, in May, 1978; received B.S. in Computer Science from North-Eastern University, Shenyang, China; received M.S. in Computer Science from East China Normal University, Shanghai, China. His research interests include formal method and software engineering.

He worked as a software engineer in Avant, SVA and DBtel Corporation from July, 2001 to July July, 2005 in Shanghai, China. Presently he works as a PH. D. candidate in Computer Science from East China Normal University, Shanghai, China. His publications include:

[17] Yefei Zhao, YANG Zong-yuan, Jinkui Xie. Formal semantics of UML state diagram and automatic verification Based on Kripke structure. 22nd IEEE

Canadian Conference on Electrical and Computer Engineering (CCECE 2009). May, 2009.

- [18] Yefei Zhao, YANG Zong-yuan, Jinkui Xie. Pi-calculus based assembly mechanism of UML state diagram and Validation of model refinement. International Conference on Electronic Computer Technology (ICECT 2009). February, 2009.

Mr. Zhao is IEEE student member. His work was supported by PhD Program Scholarship Fund of ECNU 2007.

YANG Zong-yuan was born in August, 1953, Shanghai, China. He is a professor and PH. D. supervisor in Computer Science of East China Normal University.

His research interests include software design and method, software component and formal method.

Jinkui Xie was born in October, 1975, Guilin, China. He received B.S., M.S. and PH.D. in Shanghai Jiao Tong University. Presently he works as a teacher in Computer Science, East China Normal University. His research interests include software security, trustable computation and type theory.

Qiang Liu was born in September, 1983 in Hunan province, China. Presently he is a PH. D. candidate in Computer Science, East China Normal University. His research interests include software engineering and formal method.