Journal of Computer Science 3 (6): 449-453, 2007
ISSN 1549-3636
© 2007 Science Publications

Formal Validation of the Safety Property of Sack Protocol Using Theorem Proving Technique

Shukur Z., Alias N, Mohamed Halip M.H. and Idrus B.
Fakulti Teknologi dan Sains Maklumat, Universiti Kebangsaan Malaysia
43600 Bangi, Selangor, MALAYSIA

Abstract: This paper demonstrates the formal validation process of safety properties of Selective ACKnowledgment (SACK) protocol. SACK is a complex communication protocol as it is used in various types of distributed computer systems and networks. This acknowledgment mechanism is used with sliding window protocol that allows the receiver to acknowledge packets received out of order, but within the correct sliding window. One of the critical property of SACK is its' safety property. In order to validate this property formally by using the Z/Eves theorem prover, we specify the SACK protocol using Z formal specification language. By using theorem prover tool, it helps to reduce time, energy and mistake than in relatively manual theorem proving which can be tedious and error-prone task.

Key words: formal validation, Z specification, safety property, protocol communication, SACK

INTRODUCTION

Communication protocol is a complex protocol and it is used in many distributed system and networks. Non-formal techniques are successfully used to design the protocol, but it contains unexpected error and unwanted behavior[1]. Validation needs to be done on a formal specification so that the unexpected error and unwanted behavior are discovered in the earlier development phase in order to design the correct protocol.

One of the two properties that are normally discussed in protocol communication are; safety. The other one is liveness. Safety properties are assertions that certain undesirable things do not happen[2]. For example in communication protocol, the stream of messages received should be the same as the stream transmitted, without loss, replication or permutation. Klapuri et al [3] said in their paper that a natural way of asserting a safety property is to specify the allowed initial states and state transitions.

Formal proving is a complete argument of mathematical representation and it is used to validate statement about system description. Formal proving can be done manually or with the support of formal method tools[5] such as theorem prover tools[6] such as proof checker[7]. Theorem prover is a tool that implements automatic theorem proving without the need of user support [5]. Manual proving using humans is a long and looping process and there is a great possibility a mistake will be made. Therefore in order for humans to check proofs efficiently the proofs should not be unreasonably large and they should be presented in a user-friendly fashion. However, much of the proof involved in software verification is naturally detailed, low-level and repetitious, and often results in large proofs - in short it is unsuitable for human checking. Thus, formal proving supported by tool may not only reduce the possibility of mistake but not totally removes it [8]. Therefore, the use of support tool is a main factor that can effect the acceptance of formal method practically [4]. In this research, one theorem-proving tool is chosen to support formal proving process, which is Z/EVES[9]. Z/EVES is chosen as a support tool because it can be applied in most process and need only a minimum background education such as Degree to use it. It can be learned in a few months depending on the type of applications and can be run in many platforms such as Unix, SunOS, Linux and Windows. In this research, the chosen formal specification to be developed is the specification for Selective ACKnowledgment (SACK) which is a part of TCP (Transmission Control Protocol) communication protocol. TCP is beyond the scope of this paper.

Overview of SACK: Transmission Control Protocol (TCP) acknowledgement system is a reliable sliding window transport protocol and flow control mechanism used on the Internet today. Selective ACKnowledgement (SACK) is a newer mechanism that allows the receiver to let the sender know what packet has been received. SACK is used to report multiple lost segments. In SACK, a window of TCP segment may be sent and received before an acknowledgement is received by the sender. Sender, receiver and channel are the main basic components in SACK mechanism. The flow control works when the sender first receives a message from the user application process. The message is then put in the transmission's buffer at the sender. The message is segmented and a unique

Corresponding Author: Zarina Shukur, Computer Science Dept., FTSM, University Kebangsaan Malaysia, 43600, Bangi, MALAYSIA

sequence number is attached to every segment before it is sent to the receiver through a channel. The receiver buffer the received data segment at the receiver. To validate the received segments, the receiver transmits ACK to the sender with a sequence number for the next segment it is waiting to receive. If the receiver does not receive the data segment, it will asks the sender to make a retransmission of that segment. This provides reliability, as the sender retransmits any segments that are not acknowledged by the receiver. Smith & Ramakrishnan[10] model the components, which consist of a sender, a receiver and two channels using I/O automata method as shown in Figure 1. Basic structure for the formal model described in the figure includes a sender, a receiver, a channel for packets from the sender to the receiver, and a channel for packets from the receiver to the sender. All these component is presented by the symbol S, R, C_{sr} dan C_{rs}.

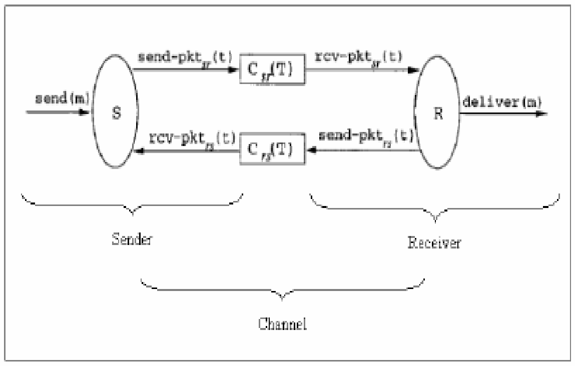

Fig 1: Structure of the SACK formal model shows the four basic components

In this paper, we will only discuss about the formal validation of Z specification of SACK's sender automata model.

SACK Sender: As described in Section 1, this paper will only discuss about the formal validation of Z specification for the sender automata model. Figure 1 shows that the sender has two input operations which are send(m) and rcv-pkt(t). The figure also shows that the sender has one output operation that is send-pkt(t). However, in detail implementation of SACK mechanism,[10] identify three input operations, three internal operations and one output operation for the sender. Thus, there are seven operations for the sender as describe below:

1. input operation that receives message from user application, send(m).
2. input operation that receives validation from the receiver upon reception of the data segment, rcv-pkt(t).
3. input operation that receives validation from the receiver upon reception of the data segment and ask for a retransmission of a data segment, rcv-pkt(ack, b1, b2, b3).
4. internal operation that prepares a data segment to be sent to the receiver, prepare-new-seg(s).
5. internal operation that prepares a retransmission data segment to be sent to the receiver, prepare-retran-seg(s).
6. internal operation that causes a state of a data segment in retransmission buffer to be set to not yet received by the receiver. This operation is enabled if time for retransmission is expired, reset-sack.
7. output operation that sends a data segment to the receiver, send-pkt(t).

All the seven operations are presented in Figure 2 as follows.

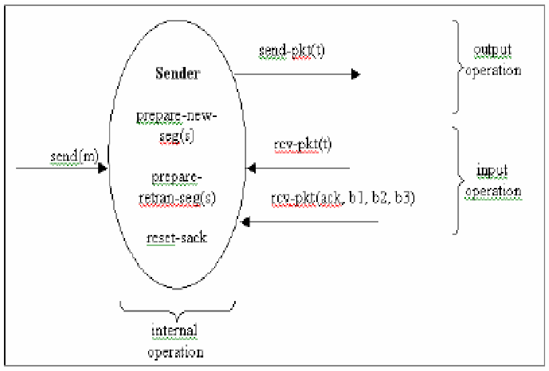

Fig. 2: Input, internal and output operation for sender

Safety property of SACK sender protocol: In 2002, Smith and Ramakrishnan[10] have developed a formal specification of TCP SACK by using I/O automaton model. They verified the safety properties by using invariant assertion and simulation (refinement) techniques. To carry out the simulation, they defined a simple automaton to represent the safety properties, which they called ReliableQ. Then, in order to prove the safety properties, they prove that a mapping from states of SACK to states of ReliableQ is a refinement mapping.

Based on the I/O automaton model by Smith and Ramakrishnan, we develop a formal specification of SACK by using a Z specification language. Before proving the safety properties, we define several invariants as proposed by Smith and Ramakrishnan in the form of theorems. These theorems are then proved by using Z/Eves theorem prover. For the safety properties, a number of theorems are developed based on the related operations and are also proved by using Z/Eves theorem prover.

The following sections discuss the specification of SACK, then the theorems that have been developed to represents the invariants and safety properties.

Z Specification of Selective Acknowledgement Protocol: Z specification of SACK sender declares variables that are used in the sender automata model into a form of paragraphs. State, initial state and operation of the sender is declared into a number of schemas. There are seven operation schemas in the Z specification for SACK sender: send, prepareNewSeg, sendPkt, rcvPkt, rcvPkt1, prepareRetranSeg and resetSack. The following section presents the specification.

Global Variables: Sender has a global variable which presents a message, sequence number of a data and some other variables. The message received from a user is hold into a buffer. The buffer is presented as a BYTE variable value 0 or 1. The message is segmented into several segments. The size of a segment is a constant value and is presented as a MSS. In this specification, we assume that one segment can have a size of an alphabet data. A ByteInt variable represents a data segment. Each segment is presented by Byte variable and a sequence number of the segment is presented by Seqnum variable. Value of Seqnum is based on the value of WS, which is a window size in the sender. The window size is fixed with a constant value of 8. Thus, a value of Seqnum is between 0 and 7. Sender also has a retransmission buffer which contains data segments, sequence number of the data segment and a state of the data segment. This is presented by SByte. BOOL variable shows the state of a data segment, either the data segment has been received or not by the receiver. If the data segment is received by the receiver, the value of the BOOL is set to TRUE. Otherwise, the value is set to FALSE. Blk variable shows a sequence number of a data segment place on the left and right of a sequence number of retransmission data segment.

```
BYTE ::= zero | one
WS == 8
MSS == 536
Seqnum == 0 .. 7
Right == Seqnum
Left == Seqnum
BOOL ::= TRUE | FALSE
ByteInt == BYTE × Seqnum
SByte == BYTE × Seqnum × BOOL
Blk == Left × Right
```

State of the Sender : State for the sender is represented by variables called state variables. The state variables are as follows:

- sendBuf represents a sender buffer which contain messages sent from user application.
- segmen presents a segmented messages.
- retranBuf variable represents a retransmission buffer
- readyToSend represents a state of the sender whether it is ready to send data or not.
- sndUna represents the sequence number of data segment which is not yet validated its reception by the receiver.
- sndNxt represents the next sequence number of data segment to be transmitted.

All these variables are declared in state schema as follows:

```
Sender
sendBuf: seq BYTE
retranBuf: seq SByte
segmen: seq ByteInt
readyToSend: BOOL
sndUna: Seqnum
sndNxt: Seqnum
```

In order to simplify the mathematical statements in the specification, five auxiliary variables are introduced, as in schema OriginalMessage. These variables are used to store the original information of the respective message.

```
OriginalMessage
message: seq BYTE
senderData: seq BYTE
receiverData: seq BYTE
retransmitData: seq BYTE
dataSegmen: seq BYTE
```

Initial State of the Sender : An initial state for the sender needs to be declared by identifying the initial value for every variable in the state schema. This is presented in schema InitSender.

```
InitSender
Sender'
-----------------
sendBuf' = ⟨⟩
retranBuf' = ⟨⟩
segmen' = ⟨⟩
readyToSend' = FALSE
sndUna' = 0
sndNxt' = 0
```

Operations of SACK' Sender: Seven schemas have been developed based on the operation for the sender as described in Section 2. All the schemas cause changes to state schema of the sender which are presented by expression DSender. Only related schemas are presented in this paper.

- send schema; the sender receives message from the user application.
- prepareNewSeg schema; It shows the sender prepares a segment that is needed to be transmitted to the receiver.
- sendPkt schema; In this operation, the sender transmits the data segment to the receiver.

- rcvPkt schema; It shows the sender has received a validated sequence number that was received by the receiver.
- rcvPkt1 schema; It shows that the sender receives a validated sequence number that was received by the receiver and need to be retransmitted.
- prepareRetranSeg schema; It shows that the sender is preparing a segment that need to be retransmitted to the receiver.
- resetSack schema; It shows that all of the segment state in the retransmission buffer is not yet received by the receiver.

```
__send______________________________________
| ΔSender
| ΔOriginalMessage
| m?: seq BYTE
|___________________
| sendBuf' = sendBuf ⁀ m?
| retranBuf' = retranBuf
| segmen' = segmen
| readyToSend' = readyToSend
| sndUna' = sndUna
| sndNxt' = sndNxt
| message' = m?
| senderData' = sendBuf'
| receiverData' = receiverData
| retransmitData' = retransmitData
| dataSegmen' = dataSegmen
|___________________________________________
```

```
__prepareNewSeg_____________________________
| ΔSender
| ΔOriginalMessage
|___________________
| readyToSend = FALSE
| sendBuf ≠ ⟨⟩
| sndNxt < sndUna + WS
| if sndNxt + 1 > 7 then sndNxt' = 0 else sndNxt' = sndNxt + 1
| sendBuf' = tail sendBuf
| segmen' = segmen ⁀ ⟨(head sendBuf, sndNxt)⟩
| retranBuf' = retranBuf ⁀ ⟨(head sendBuf, sndNxt, FALSE)⟩
| readyToSend' = TRUE
| sndUna' = sndUna
| retransmitData' = retransmitData ⁀ ⟨head sendBuf⟩
| dataSegmen' = dataSegmen ⁀ ⟨head sendBuf⟩
| receiverData' = receiverData
| senderData' = senderData
| message' = message
|___________________________________________
```

```
__rcvPkt____________________________________
| ΔSender
| ΔOriginalMessage
| ack?: Seqnum
|___________________
| retranBuf ≠ ⟨⟩
| if sndUna < ack? ≤ sndNxt
| then retranBuf' = tail retranBuf ∧ retransmitData' = tail retransmitData
| else retranBuf' = retranBuf ∧ retransmitData' = retransmitData
| readyToSend' = readyToSend
| sendBuf' = sendBuf
| segmen' = segmen
| sndUna' = ack?
| receiverData' = receiverData
| senderData' = senderData
| message' = message
| dataSegmen' = dataSegmen
| sndNxt' = sndNxt
|___________________________________________
```

Formal Validation of Safety Property of SACK Protocol: In our works, our objective is to prove that our Z specification contains the safety properties. This means that the invariants discussed in Smith and Ramakrishnan work will be one of the aspects to be proved. The second aspect that will be proved is about the operations. The invariants mentioned in [10] were defined separately from their specification. In Z, invariants can be defined in the state schemas. However, we define the invariants as theorems to make our specifications more readable.

Invariants: The four invariants (which we will call theorems after this) that will be proved are about the relation among the state variables.

- First theorem; theorem that shows "the relationship between sndUna, sndNxt and the sequence numbers of the first and last elements of retransmission buffer, retranbuf":
- If the retransmission buffer is not empty, sndUna is equal to the value of the sequence number of the first data in the buffer.

```
theorem Invariant1a
  ∀ InitSender; n: seq SByte | n = retranBuf • n ≠ ⟨⟩ ⇒ sndUna' = (head n).2
```

If the retransmission buffer is not empty, sndNxt is equal to the value of the sequence number of the last data in the buffer plus 1.

```
theorem Invariant1b
  ∀ InitSender; n: seq SByte | n = retranBuf
    • n ≠ ⟨⟩ ⇒ sndNxt' = (last n).2 + 1
```

If the retransmission buffer is not empty, sndUna is equal to sndNxt.

```
theorem Invariant1c
  ∀ Sender • retranBuf = ⟨⟩ ⇒ sndUna = sndNxt
```

- Second theorem: theorem that states "the elements of the retransmission buffer, of segments from the sender, and of the receive buffer, are sorted in ascending sequence number order. Additionally, the sequence numbers of elements of the retransmission buffer and of segments are contiguous"
- The elements of the retransmission buffer, retranBuf, are sorted in ascending sequence number order.

```
theorem Invariant2a
  ∀ Sender; i: ℕ | i ∈ dom ⟨retranBuf⟩
    • let m == retranBuf • m ≠ ⟨⟩ ⇒ (m i).2 = i - 1
```

The elements of segments from the sender, segment, are sorted in ascending sequence number order.

```
theorem Invariant2b
  ∀ Sender; i: ℕ | i ∈ dom ⟨segment⟩
    • ∃₁ n: seq ByteInt | n = segment • n ≠ ⟨⟩ ⇒ second (n i) = i - 1
```

Additionally, the sequence numbers of elements of retransmission buffer, retranBuf, and the sequence numbers of elements of segments, segment, are contiguous.

- Third theorem: states that "elements in buffers or in segments that have the same sequence number part also have the same data part".

theorem *Invariant 3*
$\forall Sender; n: SByte; q: ByteInt \mid n \in \mathrm{ran}\ retranBuf \land y \in \mathrm{ran}\ segmen$
$\bullet\ n.2 = y.2 \Rightarrow n.1 = y.1$

Proof of operations: This section describes about the theorems that represents tha safety properties of operations in SACK. The first property is that; all buffers in SACK specification should starts with an empty buffer. This property can be proved by using initial state theorem.

theorem *TheInitSender*
$\exists Sender' \bullet InitSender$

a) Safety property of sending operation; send

The message that is going to be sent, m, will be added at the back of sender buffer.

theorem *OperationSend*
$\forall send; m?: \mathrm{seq}\ BYTE \bullet sendBuf' = sendBuf ^\frown m?$

b) Safety property of preparing new segment; prepareNewSeg.

The message m, that is retrieved from the sender buffer (sendBuf) will be added at the back of retransmission buffer, retranBuf.

theorem *OperationPrepareNewSeg*
$\forall prepareNewSeg \bullet retranBuf' = retranBuf ^\frown \langle (head\ sendBuf, sndNxt, FALSE) \rangle$

c) Safety property of receiving packet; rcvPkt.

The message, m, that is removed from the retransmission buffer, is a message that is located at the front of the buffer. The message has a sequence number less than ack.

theorem *OperationrcvPkt*
$\forall rcvPkt \bullet ack? > sndUna \land ack? \leqslant sndNxt \Rightarrow retranBuf' = tail\ retranBuf$

For safety property of sending packet; sendPkt, safety property of preparing retransmission segment; prepareRetranSeg and safety property of resetting the SACK; resetSack, no theorems were developed. This is because the respective operations do not change the value of sender buffer, sendBuf (that stores message from user application) and segmen (that store the segmented messages) of sender of SACK.

All of the theorems discussed in this sections have been proved using Z/Eves thereom prover by using only one command; prove by reduce. This shows that the Z specification of SACK's Sender is reliable and contains safety properties.

CONCLUSION

In this paper, we demonstrate a validation on Z formal specification of SACK sender using theorem proving technique. According to our experience, many theorems have been through a long and repetitious proving process. If the proving is done manually by humans, the possibility a mistake will be made is higher. With Z/EVES, not only this possibility can be reduced, the proving can be done fast and reliable.

REFERENCES

1. Bochmann G.V. and C.A. Sunshine, 1983. A Survey of Formal Methods in Computer Network and Architectures and Protocols. IBM Corporation Yorktown Heights, New York. Plenum Press.
2. Duke R. and G. Rose, 2000. Formal Object-Oriented Specification Using Object-Z. MacMillan Press Ltd.
3. Klapuri H., J. Takala and J. Saarinen, 2001. Implementing reactive closed-system specifications. Journal of Network and Computer Applications., 24: 101–123
4. Babich F. and L. Deotto, 2002. Formal Methods for Specification and Analysis of Communication Protocols. IEEE Communications Surveys & Tutorials., 4(1): 2-15.
5. WetStone Technologies, Inc. October 26, 1999. Formal Methods Framework, F30602-99-C-0166, Final Monthly Status Report. Air Force Research Laboratory/IFGB, Rome, NY 13441-4505.
6. Wing, J.M., 1990. A Specifier's Introduction to Formal Methods. Computer. IEEE, 23(9): 8-23.
7. Azurat A., I.S.W.B., 2002. A Survey on Embedding Programming Logics in Theorem Prover. (online) http://www.library.uu.nl/digiarchief/dip/dispute/2002-0308-131854/2002-007.pdf.
8. Bowen J.P. and M.G. Hinchey, 1995. Ten Commandments of Formal Methods. Computer, 28(4): 56-63.
9. Meisels I. and M. Saaltink, 1997. The Z/EVES Reference Manual (for Version 1.5). Ora Canada. Canada.
10. Smith M.A. and K.K. Ramakrishnan, 2002. Formal Specification and Verification of Safety and Performance of TCP Selective Acknowledgment. IEEE/ACM Transaction On Networking, 10(2): 193-207.