International Journal of Information Technology and Knowledge Management
January-June 2011, Volume 4, No. 1, pp. 15-20

# SPECIFICATION AND VERIFICATION OF SLIDING WINDOW PROTOCOL USING PREDICATE LOGIC

Sachin Kumar[1], Veena Bharti[2] & Shikha Pandey[3]

A number of protocol verification reduction techniques were proposed in the past. Most of these techniques are suitable for verifying communicating protocols specified in the Communicating Finite State Machine (CFSM) model. However, it is impossible to formally specify communicating protocols with variables using the CFSM model. Also these methods suffered from the problem of state exploration. In this paper we have proposed a technique that used the predicate logic for specification and verification of sliding window protocol. This method is based on the technique that between sender and receiver an action is defined for any transition rule also guards and post-conditions respectively corresponding to that transition rule. On that basis safety property, deadlock freedom, Liveness property, Livelock freedom is verified.

Keywords: Sliding Window Protocol, Protocol Verification, Predicate Logic

## 1. Introduction

The design of communication protocols for computer networks [5] remains an arcane art with occasional unexpected results even in the hands of the most respected practitioners. While analysis of protocol efficiency has met with some success, verification of protocol reliability is still in its intancy. Protocol verification presupposes a clear definition of protocol performance goals or the capabilities to be provided by the protocol to its users. For data transfer, performance goals include avoiding loss, duplication or damage of messages transmitted, and delivering them in the proper sequence. For control functions, reliability goals involve the proper initialization and synchronization of control information on both sides of connection. The possibility of deadlock and the consequences of protocol failures must also be considered in assessing protocol reliability.

### 1.1. About Protocol

Communication protocols are rules whereby meaningful communication can be exchanged between different communicating entities. In general, they are complex and difficult to design and implement. Specifications of communication protocols written [6] in a natural language may be unclear or unambiguous and may be subject to different interpretations. As a result independent implementation of the same protocol may be incompatible. In addition, the complexity of the protocols make them very hard to analyze in an informal way. There is therefore a need for precise practice and unambiguous specifications using some formal languages. Many protocol implementations used in the field have almost suffered from failures, such as deadlocks when the conditions in which the protocol work correctly have been changed, there has been no general method available for determining how they will work under the new conditions, It is necessary for protocol designers to have techniques and tools to detect errors in the early phase of the design, because the later in the process that a fault is discovered, the greater the cost of rectifying it.

### 1.2. Protocol Verification

Protocols can be verified against their design and implementation. In other words, protocol can be verified either during the design phase[3] before the system is implemented, or during the testing and simulation phase after the system has been implemented. Since design verification can detect design errors at the early stage, unnecessary or incorrect implementation can be avoided. Therefore, design verification has the potential to significantly reduce the cost of protocol development and testing.

With respect to design verification, the work can be divided into two tasks: service-specification verification, and protocol specification verification. Since service specifications vary from system to system, different procedures are required for different protocols. Therefore, the protocol-specification verification has been the focus of most of researchers[3].

In summary, protocol verification is a task which attempts to detect the existence of logic errors in the protocol design specification at the early development stage.

[1, 2, 3]R.K.G.I.T, Ghaziabad

Email: [1]imsachingupta@rediffmail.com, [2]bharti.veena@gmail.com, [3]shikpan@gmail.com

### 1.3. Predicate Logic

The predicate calculus is fundamental in mathematical logic; important restrictions: the only allowed symbols are $\rightarrow$, $\perp$, $\forall$ respectively read as "implies", "false", "for all". In fact, every other logical symbols can be defined with them. This restriction is therefore only syntactic, but not semantic.

We suppose given an infinite set of variables: {x,y,….}, an infinite set of constants: c = {a,b,….}and some predicate symbols P, Q, R,….; each of them has an arity which is an integer $\geq 0$.

Atomic formulas are of the form $\perp$ (read false) or Pt1….Ptk (denoted as p(t1,…,tk)) where P is a predicate symbol of arity k and t1,….,tk are variables or constants.

Formulas of the predicate calculus are built with the following rules:

- An atomic formula is a formula.
- If F and G are formulas, then F $\rightarrow$ G is a formula(read "F implies G").
- If F is a formula and x is a variable, then $\forall$x F is formula(read " for all x, F").

## 2. FORMAL SPECIFICATION OF SLIDING WINDOW PROTOCOL

In sliding window protocol [15], each outbound frame contains a sequence number, ranging from 0 up to some maximum. At any instant of time, the sender maintains a fixed size buffer corresponding to set of frames, sent but yet not acknowledged, with their sequence numbers. This buffer is termed sender's window. This storage is done for possible retransmission; since sent frames may ultimately be lost or damaged in transit. Since it has multiple outstanding frames, it maintains multiple logical timers, one per outstanding frame. Each frame times out independently of all the other once. Similarly the receiver also maintains a receiver's window corresponding to the set of frames it is permitted to accept. The receiver has a buffer reserved for each sequence number within its window. Associated with each buffer is a bit telling whether the buffer is full or empty? Whenever a frame arrives, its sequence number is checked to see if it falls within the window. If so, and if it has not already been received, it is accepted and stored. An acknowledgement is sent also, if its predecessor frame has been acknowledged. Whenever the receiver has reason to suspect that an error has occurred, it sends a negative acknowledgement (NAK) frame back to the sender. Such a frame is a request for retransmission of the frame specified in the NAK. The Sender retransmits a frame, either on receiving NAK or on being timed out, whichever is earlier. Any frame falling outside the window is discarded without comment. The sender's window and receiver's window need not have the same lower and upper limits, or even have the same size. A frame buffer is released if buffer for its predecessor frame has already been released. Sender releases buffer after receiving acknowledgement while receiver does the same after sending acknowledgement. We define the states for different processes of the system as follows.

### 2.1. Sender and Receiver

In a SW protocol, there are two main components: the sender and the receiver. The sender obtains an infinite sequence of data from the sending host. We call indivisible blocks of data in this sequence "frames", and the sequence itself the "input sequence". The input sequence must be transmitted to the receiver via an unreliable network. After receiving the frames, the receiver eventually delivers them to the receiving host. The correctness condition for a SW protocol says that the receiver should deliver the frames to the receiving host in the same order in which they appear in the input sequence.

### 2.2. Messages and Channels

In order to transmit a frame, the sender puts it into a frame message together with some additional information, and sends it to the frame channel. After the receiver eventually receives the frame message from this channel, it sends an acknowledgment message for the corresponding frame back to the sender. This acknowledgment message is transmitted via the acknowledgment channel. After receiving an acknowledgment message, the sender knows that the corresponding frame has been received by the receiver. Thus the communication between the sender and the receiver is bi-directional; the sender transmits frames to the receiver via the frame channel, and the receiver transmits acknowledgments for these frames to the sender via the acknowledgment channel.

### 2.3. Sequence Numbers

The sender sends the frames in the same order in which they appear in its input sequence. However, the frame channel is unreliable, so the receiver may receive these frames in a very different order (if receive at all). Therefore it is clear that each frame message must contain some information about the order of the corresponding frame in the input sequence. Such additional information is called "sequence number". If we include as a sequence number the exact position of the frame in the input sequence, it would make sequence numbers used by our protocol unbounded (because conceptually the input sequence is infinite). Since unbounded sequence numbers are not practical. This is why in a SW protocol, instead of the exact position of the frame in the input sequence, the sender sends the remainder of this position with respect to some fixed modulus K. The value of K varies greatly among protocols: it is only 16 for the Mascara protocol for wireless ATM networks, but 232 for TCP. To acknowledge a frame, the receiver sends in the acknowledgment message the sequence number for which the frame was received. It should be noted that acknowledgments are "accumulative"; for example, when the sender acknowledges a frame with sequence number 3, it means that frames with sequence numbers 0, 1 and 2 have also been received.

### 2.4. Sending and Receiving Windows

At any time, the sender maintains a sequence of sequence numbers corresponding to frames it is permitted to send. These frames are said to be a part of the sending window. Similarly, the receiver maintains a receiving window of sequence numbers it is permitted to receive. In our protocol, the sizes of sending and receiving windows are equal and represented by an arbitrary integer N. At some point during the execution it is possible that some frames in the beginning of the sending window are already sent, but not yet acknowledged, and the remaining frames are not sent yet. When an acknowledgment arrives for a frame in the sending window that is already sent, this frame and all preceding frames are removed from the window as acknowledgments are accumulative. Simultaneously, the window is shifted forward, such that it again contains N frames. As a result, more frames can be sent either immediately or later. Acknowledgments that fall outside the window are discarded. If a sent frame is not acknowledged for a long time, it usually means that either this frame or an acknowledgment for it has been lost. To ensure the progress of the protocol, such frame is eventually resent. Many different policies for sending and resending of frames exist [22], which take into account, e.g., the efficient allocation of resources and the need to avoid network congestion. Here we are only concerned with the correctness of the protocol, so we abstract from the details of the transmission policy and specify only those restrictions on protocol's behavior that are needed to ensure safety. During the execution, the receiving window is usually a mix of sequence numbers corresponding to frames that have been received out of order and sequence numbers corresponding to "empty spaces", i.e. frames that are still expected. When a frame arrives with a sequence number corresponding to some empty space, it is inserted in the window, otherwise it is discarded. At any time, if the first element of the receiving window is a frame, it can be delivered to the receiving host, and the window is shifted by one. The sequence number of the last delivered frame can be sent back to the sender to acknowledge the frame (for convenience reasons, we acknowledge delivered frames instead of received frames). It should be noted that not every frame must be acknowledged; it is possible to deliver a few frames in a row and then acknowledge only the last of them.

## 3. Verification of Sliding Window Protocol

Each process executes at nonzero speed but we make no assumption on the relative speed of processes. Several CPUs may be present but memory hardware prevents simultaneous access to the same memory location. We also make no assumption about order of interleaved execution. Almost every model used for correctness analysis assumes that the execution of a concurrent system can be viewed in terms of events that can be considered atomic [16]. Our technique also views the execution of the system in terms of the atomic events. These events are communication with the other process, that is, a message transfer. Due to our assumption regarding atomicity, we can formalize a data link layer protocol as a state transition system. We model the system by considering the presence of a set of processes in the system, namely

1. SP is a sender process which sends frames in to the channel.
2. RP is a receiver process which receives frames from the channel.
3. Fra_OK is a flag, which shows that incoming frame does not contain any error.
4. Timer process.
5. W_rec is a flag that shows the receiver process has received all frames of the current window.
6. PP is a producer process which delivers frames to the sender process.

The predicate expression in(P.x) represents that a process P is in state P.x A state transition rule represents the movement of process from one state to other. For firing any transition rule P.r there exist a corresponding weakest precondition wp(P.r, R). If the system state satisfies the condition wp(P.r, R) then execution of the transition rule P.r will eventually establish the post condition R. We define an operator "leads to" (symbolized as "$\rightarrow$") in wp environment with the following semantics. $Q \rightarrow R$ implies

$$\exists r : wp(r, R) = Q \qquad (1)$$

where Q and R are guards and post-conditions respectively corresponding to transition rule r.

### 3.1. Sender Process SP have Following States

| State | Meaning |
|---|---|
| 1. SP. ready_a | SP is ready to send frame_a |
| 2. SP. sent_a | SP has sent frame_a |
| 3. SP. recack_a | SP has received acknowledgement for frame_a |
| 4. SP. resend_a | SP resends frame_a and restarts its local Timer |
| 5. SP. nrecak_a | SP has received negative acknowledgement for frame_a |

| | |
|---|---|
| 6. SP. relbuf_a | SP has released buffer occupied by frame_a |

### 3.2. Receiver Process RP have Following States

| State | Meaning |
|---|---|
| 1. RP. ready_a | RP is ready to receive frame_a |
| 2. RP. rec_a | RP has received frame_a |
| 3. RP . sentack_a | RP has sent acknowledgement for frame_a |
| 4. RP . nsentak_a | RP has sent negative acknowledgement for frame_a |
| 5. RP . inbuf_a | RP has put frame_a in buffer |
| 6. RP . relbuf_a | RP has released buffer occupied by frame_a |
| 7. RP. naccept_a | RP will not accept frame_a |

### 3.3. Fra_Ok Flag have Following States

| State | Meaning |
|---|---|
| 1. Fra_Ok=true | If the frame arrived at the receiver does not have any error |
| 2. Fra_Ok=false | If the frame arrived at the receiver has some error |

### 3.4. Timer Process have Following States

| State | Meaning |
|---|---|
| 1. Tim_idle_a | Timer for frame_a is idle |
| 2. Tim_start_a | Timer for frame_a starts |
| 3. Tim_end_a | Timer for frame_a stops |
| 4. Tim_restart_a | Timer for frame_a restarts |

### 3.5. W_rec Flag have Following States

| State | Meaning |
|---|---|
| 1. W_rec=false | All frames related to current window have not been received, except frame a. |
| 2. W_rec=true | All frames related to current window have been received, except frame a. |

### 3.6. Producer Process PP have Following States

| State | Meaning |
|---|---|
| 1. PP . idle | Producer is idle |
| 2. PP . produced_a | Producer has delivered frame a to sender process |

The processes on the basis of above states are as follows:

Process SP: initialized as in(SP. ready_a)

Transition rule

Let,

Ws = size of sender's window

a = frame in sender's window where a will assume values 0, 1, …, (Ws – 1) in sequence

b = next frame in sender's window such that b = (a + 1) mod Ws

c = arbitrary frame in sender's window

a' = first frame in sender's window after sliding such that

a' = c + 1

The range of values assumed by a' is same as a.

SP . send_frame

def R1= in(SP . ready _b) $\wedge$ in(SP . sent_a) $\wedge$ in(Tim_start_a)

$\wedge$ in(Channel_launched)

R2 = in(SP . relbuf _a) $\wedge$ in(Tim_end_a)

$\wedge$ in(SP . ready _b) $\wedge$ in(SP . recack _a)

R3=in(SP . resend _a) $\wedge$ in(Tim_restart_a) $\wedge$ in(SP . ready_c)

$\wedge$ in(SP . nsentak _a)

R4=in(SP . resend _a) $\wedge$ in(Timer_restart_a) $\wedge$ in(SP . ready_c)

R5=in(SP . relbuf _a-to-c) $\wedge$ in(SP . ready _ a')

R = R1 $\vee$ R2 $\vee$ R3 $\vee$ R4 $\vee$ R5

Q1=in(SP . ready _a) $\wedge$ in(PP . produced_a)

Q2=in(SP . sent_a) $\wedge$ in(Tim_start_a)

$\wedge$ in(Channel_produce_ack_i)

Q3=in(SP . ready _c) $\wedge$ in(Channel_produce_nak_a)

Q4=in(SP . ready _c) $\wedge$ in(Tim_end_a)

Q5=in(SP . resend _a) $\wedge$ in(Channel_produce_ack_c)

wp(SP . send_frame, R) =(Q1 $\vee$ Q2 $\vee$ Q3 $\vee$ Q4 $\vee$ Q5)

$\wedge$ (Q1 $\Rightarrow$ wr(select, in(SP . send_frame . s1)))

$\wedge$ (Q2 $\Rightarrow$ wr(select, in(SP . send_frame . s2)))

$\wedge$ (Q3 $\Rightarrow$ wr(select, in(SP . send_frame . s3)))

$\wedge$ (Q4 $\Rightarrow$ wr(select, in(SP . send_frame . s4)))

$\wedge$ (Q5 $\Rightarrow$ wr(select, in(SP . send_frame . s5)))

$\wedge$ (in(SP . send_frame . s1) $\Rightarrow$ wp(SP . send_frame1, R1))

$\wedge$ (in(SP . send_frame . s2) $\Rightarrow$ wp(SP . send_frame2, R2))

$\wedge$ (in(SP . send_frame . s3) $\Rightarrow$ wp(SP . send_frame3, R3))

$\wedge$ (in(SP . send_frame . s4) $\Rightarrow$ wp(SP . send_frame4, R4))

$\wedge$ (in(SP . send_frame . s5) $\Rightarrow$ wp(SP . send_frame5, R5))

End of the transition rule SP . send_frame;

End of the process SP;

Process RP; initialized as in(RP . ready _a)

Transition rule

Let,

RW = size of receiver's window

a = frame in receiver's window where a will assume values

0, 1, …, (RW – 1) in sequence

b = next frame in receiver's window such that

b = (a + 1) mod RW

c = arbitrary frame in receiver's window

a' = first frame in receiver's window after sliding such that a' = c + 1

The range of values assumed by a2 is same as a.

RP . receive_frame

def $R_1$= in(RP. rec_a) $\wedge$ in(RP . ready _a) $\wedge$ in(RP . sentack_i)

$\wedge$ in(RP. relbuf _a) $\wedge$ in(W_rec=false)

R2=in(RP . ready _a) $\wedge$ in(RP . inbuf_c)

R3= in(RP . rec_a) $\wedge$ in(RP . ready _ a') $\wedge$ in(RP . sentack_c)

$\wedge$ in(RP . relbuf _a-to-c) $\wedge$ in(Win_rec=true)

R4=in(RP . ready _a) $\wedge$ in(RP . nsentak_a)

R5=in(RP . no_accept_c)

R=R1 $\vee$ R2 $\vee$ R3 $\vee$ R4 $\vee$ R5

Q1=in(RP . ready _a) $\wedge$ in(Channel_produce_i)

$\wedge$ in(Frame_a_Ok=true)

Q2=in(RP . ready _a) $\wedge$ in(Channel_produce_c)

$\wedge$ in(Frame_c_Ok=true) $\wedge$ $\neg$ in(RP . inbuf_c)

Q3=in(RP . ready _a) $\wedge$ in(Channel_produce_a)

$\wedge$ in(RP . inbuf_b-to-c) $\wedge$ in(Frame_a_Ok=true)

Q4=in(RP . ready _a) $\wedge$ in(Channel_produce_a)

$\wedge$ in(Frame_a_Ok=false)

Q5=in(RP . inbuf_c) $\wedge$ in(Channel_produce_c)

wp(RP . receive_frame, R) = (Q1 $\vee$ Q2 $\vee$ Q3 $\vee$ Q4 $\vee$ Q5)

$\wedge$ (Q1 $\Rightarrow$ wr(select, in(RP . receive_frame . s1)))

$\wedge$ (Q2 $\Rightarrow$ wr(select, in(RP . receive_frame . s2)))

$\wedge$ (Q3 $\Rightarrow$ wr(select, in(RP . receive_frame . s3)))

$\wedge$ (Q4 $\Rightarrow$ wr(select, in(RP . receive_frame . s4)))

$\wedge$ (Q5 $\Rightarrow$ wr(select, in(RP . receive_frame . s5)))

$\wedge$ {(in(RP . receive_frame . s1) $\Rightarrow$ wp(RP . receive_frame1, R1))}

$\wedge$ {(in(RP . receive_frame . s2) $\Rightarrow$ wp(RP . receive_frame2, R2))}

$\wedge$ {(in(RP . receive_frame . s3) $\Rightarrow$ wp(RP . receive_frame3, R3))}

$\wedge$ {(in(RP . receive_frame . s4) $\Rightarrow$ wp(RP . receive_frame4, R4))}

$\wedge$ {(in(RP . receive_frame . s5) $\Rightarrow$ wp(RP . receive_frame5, R5))}

End of the transition rule RP . receive_frame;

End of the process RP;

## 4. PROOF OF CORRECTNESS

Correctness of any protocol can be established by showing that the protocol satisfies the logical properties: safety, deadlock, and progress properties.

### 4.1. Safety Property

All frames must be received without repetition. It will be proved in two parts:

(a) Transition rule Q5 $\rightarrow$ R5 of receiver process reveals that weakest precondition for not accepted frame c is "frame c is already in buffer of receiver". Thus any frame once received will never be received again.

(b) 'All' frames must be received. We interpret, "if all frames of current window are received, only then receiver should become ready to receive first frame of next window". This condition can be represented in predicate form as follows.

in(RP . rec_a) $\wedge$ in(W_rec=true) $\rightarrow$ in(RP . ready_m) (2)

As defined previously m = a', where a' is sequence number of first frame in next window.

Transition rule Q3 $\rightarrow$ R3 exhibits that

Q3 $\rightarrow$ {in(RP . rec_a) $\wedge$ in(W_rec=true) $\wedge$ in(RP. ready_m) $\wedge$ I} (3)

Where I = in(RP . sentack_c) $\wedge$ in(RP . relbuf _a-to-c)

Also the predicate shown in Eq. (2) is weaker than that of in Eq. (3). Thus correctness of the predicate shown in Eq. (2) is ensured.

### 4.2. Deadlock Freedom

A state deadlock occurs when each and every processes can only remain indefinitely in the same state. Sender and receiver are two non-competing processes, represented through guarded commands. In guarded commands, though more than one guards can be true at a time, only one true guard will be selected and corresponding statement will be executed. Thus deadlock freedom is ensured.

### 4.3. Progress Property

We need to prove that each of the frames is incremented by one infinitely with finite delay. Transition rule Q1 $\rightarrow$ R1 of the sender process reveals that transition from ready to send frame a to ready to send frame b (where b = a + 1) needs no co-operation from any other process except it's producer process. Producer process supplies frames from upper layer to sender process. Thus, every frame supplied by producer process will eventually be transmitted.

## 5. Conclusion

In this paper, I have argued that predicate logic can be used to specify protocols in a simple and elegant manner. In particular, we have seen how to use propositional linear predicate logic to specify the protocols. Of course, using this logic to express the verification is not new, but the application of this technique to specification of protocol is both novel and we believe, very promising .In future, this method can be applied for the verification of other protocol and also new method can be deduced to verify the protocol.

## References

[1] Chung-Ming Huang; Duen-Tay Huang, "A Backward Protocol Verification Method", TENCON'93 Proceedings. Computer, Communication, Control and Power Engineering, 1993 IEEE Region 10 Conference on, 19-21 Oct. 1993, 1, Page(s) 515-518.

[2] Chung-Ming Huang; Jenq-Muh Hsu; Huei-Yang Lai; Jao-Chiang Pong; Duen-Tay Huang, "An Incremental Protocol Verification Method for ECFSM-based Protocols", Proceedings of the Eighth Annual Conference, on 14-17, June 1993, Page(s) 87–97.

[3] Yuang,M.C.; "Survey of Protocol Verification Techniques based on Finite State Machinemodels", Computer Networking Symposium, 1988., Proceedings of the 11-13 April 1988, Page(s) 164 –172.

[4] Wen-Chien Liu; Chyan-Goei Chung; "Protocol Verification using Concurrent Paths", Communications, Fifth Asia-Pacific Conference on ... and Fourth Optoelectronics and Communications Conference, 2, 18-22, Oct. 1999 Page(s) 1188 -1191.

[5] Behrouz A Forouzan, "Data Communications and Networking", Tata McGraw Hill.

[6] Andrew S. Tanenbaum, Computer Networks, 4/E, Prentice Hall, 2003.

[7] G. M. Lundy and R. C. McArthur, "Formal Modal of a High Speed Transport Protocol," in Protocol Specification, Testing and Verification XII. North-Holland, The Netherlands, 1992.

[8] Lee, T.T.; Lai, M.-Y.; "A Relational Algebraic Approach to Protocol Verification Software Engineering", IEEE Transactions, 14, Feb. 1988 Page(s) 184 – 193.

[9] D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead," IEEE Commun. Mag., June 1989.

[10] A. Netravali, W. Roome, and K. Sabnani, "Design and Implementation of a High Speed Transport Protocol." IEEE Trans. Commun., 38, Nov. 1990.

[11] G. M. Lundy and R. E. Miller, "Specification and Analysis of a Data Transfer Protocol using Systems of Communicating Machines," Disirib. Comput., Dec. 1991.

[12] El Maati Chabbar, Mohammed Bouhdadi "On Verification of Communicating Finite State Machines using Residual Languages" Proceedings of the First Asia International Conference on Modeling and Simulation(AMS, 07), 2007.

[13] Edgar Pek, Nikol Bogunovic "Predicate Abstraction in Protocol Verification", 8th International Conference on Telecommunication ConTEL 2005, June 15-17, Zagreb, Croatia.

[14] Andras L. Olah ans Sonia M. Heemstra" Alternative Specification and Verification of a Periodic State Exchange Protocol", IEEE ACM Transactions on Networking, 5, No.4, August 1997.

[15] Ulle Endriss "Temporal Logic for Representing Agent Communication Protocol".

[16] Jean-Louis Krivine and Yves Legrandgerard, "Valid Formulas, Games and Network Protocols", 14 November 2007.

[17] A.S. Tanenbaum and M. Van Steen, "Distributd Systems: Principles and Paradigms", Upper Saddle River, NJ: Prentice Hall, 2002.

[18] M. Smith and N. Klarlund, "Verification of a Sliding Window Protocol using IOA and MONA", Formal Methods for Distributed System Development, Pages19-34, Kluwer Academic Publishes, 2000.

[19] M. Fisher, "Temporal Development Methods for Agent-based Systems", Journal of Autonomous Agents and Multi-agent Systems, 10, 41-66, 2005.

[20] M. G. Gouda and N. Multari, "Stabilizing Communication Protocols," IEEE Trans. Comput., 40, pp. 448-458, Apr. 1991.

[21] G. M. Lundy and R. C. McArthur, "Formal Modal of a High Speed Transport Protocol," in Protocol Specification, Testing and VerificationXII, North-Holland, The Netherlands, 1992.

[22] Bhaskar Sardar, Debashis Saha, "A Survey of TCP Enhancements for Last-Hop Wireless Networks" 3rd Quarter 2006, 8, No. 3, www.comsoc.org/pubs/surveys.