

Modeling and Formal Verification of DHCP Using SPIN

Syed M.S. Islam¹, Mohammed H. Sqalli¹ and Sohel Khan²

¹*Department of Computer Engineering, King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia.*

²*Department of Information and Computer Science, KFUPM, Dhahran, Saudi Arabia.
sislam123,sqalli,sohel@ccse.kfupm.edu.sa*

Abstract

The Dynamic Host Configuration Protocol (DHCP) is a widely used communication protocol. In this paper, a portion of the protocol is chosen for modeling and verification, namely the assignment of new IP address to a newly arriving host. PROcess Meta LAnguage (PROMELA) is used for modeling and the verification is performed using SPIN. SPIN can verify most of the communication protocols either by performing random simulations or by generating a C program. It can perform an exhaustive verification that can establish with mathematical certainty whether or not a given behavior is error-free. PROMELA is used to specify a system behavior in a formal validation model that defines interactions of processes in a distributed system and allows for the dynamic creation of concurrent processes. We have analyzed and verified various properties of the DHCP protocol such as absence of deadlock, livelock, and improper termination under various conditions such as message loss or arbitrary errors. We obtained the expected execution of the protocol using simulation. No invalid end-state or Non-progress cycle were found. We have also formalized a linear time temporal logic (LTL) to verify whether messages are exchanged properly and whether two clients may get the same IP address at the same time. No violation of the first claim was found, but the latter was violated.

Keywords: *Communication protocol, modeling and formal verification, PROMELA, SPIN, deadlock, livelock, liner temporal logic.*

1. INTRODUCTION

Protocol verification is the process of examining the protocol specifications for the presence of various errors that could lead to improper operation. Formal verification is better than testing by implementation [1, 2] because:

- It can prove the absence of errors like deadlock or livelock (exhaustive testing).
- It is less costly to fix errors.
- It is independent of implementation.
- It is fully automated.

SPIN [3] is one of the prominent verifiers that can verify most of the communication protocols. It either performs random simulations of a protocol's execution or generates a C program that performs an efficient verification of the protocol's correctness properties. But the protocol has to be modeled in a high level language like PROcess Meta LAnguage (PROMELA). Using this language, a system behavior can be specified in a formal validation model that defines interactions of processes in a distributed system. It allows for the dynamic creation of concurrent processes [4]. SPIN can perform an exhaustive verification that can establish with mathematical certainty whether or not a given

behavior is error-free. To verify very large systems, it provides a “bit state storage” technique known as *supertrace* that can collapse the state space to a small number of bits per reachable state with minimal side-effect [4].

To be connected to the Internet, a host requires an IP address, a subnet mask and the address of a nearby router [5]. These elements must be reconfigured when a host moves or relocates to a new place. DHCP provides a framework for passing these configuration parameters to a fixed or mobile host connecting to a TCP/IP network.

DHCP evolved from the Bootstrap Protocol (BOOTP) which provided a solution to a problem in Reverse Address Resolution Protocol (RARP) [6]. By using UDP messages, BOOTP (RFCs 951, 1048, 1084, 1542) solved the problem of forwarding a client’s broadcast of its Ethernet Address (EA) and the requirement of a RARP server on each network. It also provides additional information such as the IP address of the file server holding the memory image, the IP address of the default router and the subnet mask to use. But the network administrator has to manually configure the tables mapping the IP address to the EA and for a newly arriving client, an IP address has to be assigned and an entry for this client has to be added to the configuration table before it can use the protocol. To solve the entire problem, DHCP was proposed by the Dynamic Host Configuration Working Group of the Internet Engineering Task Force (IETF). Its definition is recorded in Internet RFC 2131 [7] and the Internet Activities Board (IAB) is asserting its status as to Internet Standardization [8].

The plug and play nature [9] of DHCP, and its ability to automate the network-related aspects of connecting a host into a network, makes it very attractive to the network administrators who would otherwise have to perform these tasks manually. It enjoys a widespread use in residential Internet access networks as well as in wireless LANs, where hosts are frequently joining and leaving the network. It is also a good candidate for exploring the potential of model checking and verification due to its characteristics of concurrency and distributed computing.

SPIN and PROMELA have been used for modeling and verification of a number of communication protocols and software. A formal analysis and verification of a space-craft controller is well-described in [10]. Model checking of legacy flight software from NASA’s Deep Space One (DS1) mission using SPIN is described in [11]. Instant Messaging and Presence Protocol (IMPP) is modeled using PROMELA and verified using SPIN in [12]. These tools are used for verifying cryptographic protocols [13]. SPIN and PROMELA are also used for modeling and verifying software for a nuclear power station cooling system [14], Flexible Manufacturing System (FiMS) [15], web services [16] and embedded systems [17]. But to our knowledge, no significant work has been done for modeling DHCP using PROMELA and verification with SPIN. A portion of DHCP has been modeled using C programming language [18] as a case study of verifying network protocol implementations by Symbolic Refinement Checking between two models: the implementation extracted from the code and the specification extracted from the RFC. One client, one server and one communication link were modeled to check whether the client was implemented correctly or not. No inconsistency was reported.

The main aim of this paper is to model and verify some of the important portions of DHCP protocol. The integrity of communication between the client and server and the ability to recover from message distortions or loss is investigated. It is also checked whether there is any instance of deadlocks (states in which no further protocol execution is possible; occurring when all the protocol processes are waiting for conditions that can never be fulfilled) and livelocks (execution sequences that can be repeated indefinitely often without ever making effective progress) in the protocol. Presence of any improper termination i.e., the completion of the protocol execution without satisfying the proper termination conditions is also tested. Finally, the property that any specific IP address will not be in use by more than one DHCP client at a time is verified.

In the next section, the protocol specification is discussed in short. The proposed model of the protocol is described in section 2. Results of various verifications are summarized and discussed in section 4. Conclusion is presented in section 5.

2. PROTOCOL SPECIFICATION

Before discussing the proposed model, it is helpful to describe the specification of the DHCP protocol. In protocol engineering, a protocol is specified by the services to be provided, assumptions about the environment, vocabulary of the messages used, encoding (format) of each message, and the procedure rules for consistency of message exchanges. An overview of these five basic elements of DHCP is given in the following subsections.

2.1. SERVICES

DHCP services can be categorized into two classes: (i) Providing persistent storage of network parameters and (ii) Allocation of network addresses [7]. DHCP stores a key-value entry for each client, where the key is some unique identifier and the value contains the configuration parameters for the client. The key will be (IP-subnet-number, hardware-address) unless the client explicitly supplies an identifier using the 'client identifier' option. A client can query the DHCP service to retrieve its configuration parameters.

For the allocation of a network address, DHCP supports three mechanisms. In the “automatic allocation”, an IP address is assigned permanently. And, in the “manual allocation”, a client's IP address is assigned by the network administrator and DHCP is used simply to convey the assigned address to the client. In the “dynamic allocation”, a client requests the use of an address for some period of time. The server provides the address and guarantees not to reallocate that address within the requested time and attempts to return the same network address each time the client requests an address. The client may extend its lease with subsequent requests. In case of scarcity of IP addresses, the server reuses address whose lease has expired. The services are accomplished through the exchange of a number of messages between the client and the server(s).

2.2. ASSUMPTIONS

The following assumptions are made about the environment:

- The entities involved are: (i) DHCP client that initiates a request for network parameters (ii) DHCP server that responds to the client's requests; (iii) DHCP relay agent that acts as an intermediate entity between a DHCP client and a DHCP server. There will be one or more of these entities present in a scenario.
- The transmission channel is full duplex and may be fixed or wireless.
- The transmission channel is assumed to provide errors or losses since UDP is used as the transport layer protocol [7].
- There are some broadcast messages and some unicast messages.

2.3. VOCABOLARY

DHCP works with the following eight types of messages:

- DHCPDISCOVER: Broadcast message of a newly arriving client.
- DHCPOFFER: Server to client message with an offer of configuration parameters.
- DHCPREQUEST: Client's message to servers after getting an offer.
- DHCP ACK: The server's response to the DHCP request message.

- DHCP NAK: Server's response when the client's IP address is incorrect or the lease is expired.
- DHCPDECLINE: Client to server message indicating the network address is already in use.
- DHCPRELEASE: Client to server relinquishing an IP address and canceling the remaining lease.
- DHCPINFORM: Client to server message asking only for local configuration parameters.

Table 1: DHCP message format

op(1)- message type	htype(1)- h/w address type	hlen(1)- h/w address length	hops(1) –used by relay agent
xid(4)-transaction ID			
secs(2)-seconds elapsed since addr acquisition or renewal			Flags(2)
ciaddr(4)-client IP address[for bound, renew or rebinding state]			
yiaddr(4)- 'your'(client) IP address			
siaddr(4)-IP address of next server to use			
giaddr(4)-relay agent IP address			
chaddr(16)-client h/w address			
sname(64)-optional server host name.			
File(128)-boot file name			
Options (variable)-optional parameters field.			

2.4. FORMAT

The encoding or formatting of each message as described in RFC [7] is summarized in the Table 1.

2.5. PROCEDURE RULES

The procedure rules involved in the assignment of new IP address are illustrated in Figure 1 [20] and summarized below:

1. A client broadcasts a DHCPDISCOVER message within a UDP datagram to port 67 using the broadcast destination address of 255.255.255.255 and a "this host" source address of 0.0.0.0. This message may include options that suggest values for the network address and lease duration. Relay agents may pass the message to DHCP servers not on the same physical subnet.
2. A DHCP server responds with a DHCPOFFER message containing the transaction ID of the received discovery message, the proposed IP address for the client, the network mask and an IP address lease time. The lease time is the amount of time for which the IP address will be valid (generally several hours or days [21]). This message is transmitted to the client, using the relay agent if necessary.
3. The client chooses one offer from one or more server offers and responds to its selected offer with a DHCPREQUEST message, echoing back the configuration parameters. This message is broadcast and relayed through relay agents. If the client receives no DHCPOFFER messages, after an ad-hoc timeout, the DHCPDISCOVER message will be retransmitted.

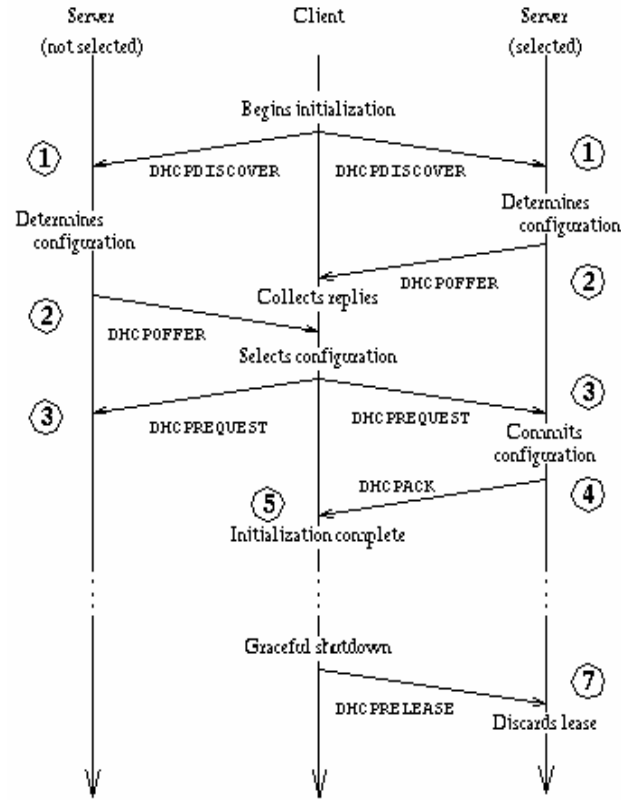


Figure 1: Procedure rules in assigning a new IP address involving two servers.

4. The selected server commits binding and responds with a DHCPACK message containing the configuration parameters for the requesting client. If it is unable to satisfy the DHCPREQUEST message, a DHCPNAK message is sent. The unselected server(s) notifies client's declination of its offered IP address if it finds a different server id parameter in the request message. The server checks whether the IP address to be sent is already assigned or not by using the Internet Control Message Protocol (ICMP) protocol.
5. Upon reception of a DHCPACK, the client performs a final check on the parameters (e.g., ARP (Address Resolution Protocol) for allocated network address), and notes the duration of the lease. At this point, the client is configured. If the address is already in use, the client sends a DHCPDECLINE message to the server and restarts the whole configuration process. It may send a DHCPRELEASE message to relinquish an IP address and cancel the remaining lease.

If the client receives a DHCPNAK message, then it restarts the configuration process. To extend the lease of the current IP address, the client has to follow the 'renew' procedure [19].

3. THE PROPOSED MODEL

The objective of modeling a protocol is to study its structure and to verify its completeness and logical consistency. Therefore, in a protocol model, functional specifications of the protocol are abstracted into a minimal working specimen that can enable us to understand and analyze a particular aspect of it more closely [11]. In modeling, simplifications are also needed for a description at an abstract level

to avoid implementation details. For example, we need not to describe how a message is encoded, transmitted, or stored. We should focus on the design of a complete and consistent set of rules for interactions in a distributed system. In the following sections, the level of abstraction and the details of the proposed model are described.

3.1. ABSTRACTION

We have only modeled the DHCP service of allocating a new IP address to a newly arriving host, because the other service, namely the storage of configuration parameters, is not related to message communication. The reuse/rebind procedure is also ignored for its heavy relationship with configuration parameter storage. The relay agent is also not modeled as it is an optional medium of communication and not related to the concurrent messages exchanged. Instead of modeling multiple entities, the presence of two servers and two clients is modeled, which is enough for describing the expected behavior of the protocol. The message format is abstracted with the following fields only: Op, xid, siaddr, yiaddr (see Table 1). All the messages are considered, excluding DHCPINFORM as this is not related to communication, but rather to storage or configuration.

3.2. IMPLEMENTATION DETAILS OF THE PROPOSED MODEL

A PROMELA model is constructed from three basic types of objects: Message channels, Data objects and Processes [1]. Each of these components of the proposed model is described in the following subsections.

3.2.1. MESSAGE CHANNELS

For simplicity of understanding, eight one-directional channels are used to communicate among two clients and two servers in the proposed model as illustrated in Figure 2.

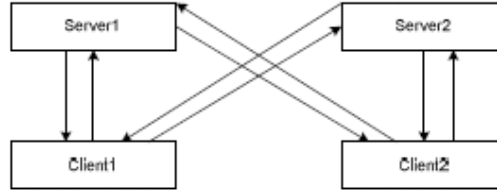


Figure 2: Communication channels between the clients and the servers.

All the channels are defined as global variables. Each channel is modeled to store one message consisting of four fields: a message type name and three values of size 8-bit each. For example, the following channel declaration in PROMELA defines the channel between Client2 to Server2 that can store a message, e.g., request message with fields sid, xid, ip.

```
chan cl2to_server2 = [1] of {mtype, byte, byte, byte};
```

As mentioned earlier, we have abstracted seven types of messages, declared as follows:

```
mtype = { disc, offer, req, dec, ack, nak, release};
```

3.2.2. DATA OBJECTS

Two dimensional arrays are used to store the IP address list of the two servers. The first column of the array contains the values while the other indicates whether the IP is available or not. Although IP addresses are of 32 bits, we have abstracted them using a byte type as follows:

```
Typedef ip2d { /*used to define a 2D array type*/
    byte el[2]; /*element that indicate availability*/
};
```

```
ip2d ip_server1[2] ; /* this is a 2D array to store 2 IP addresses of Server1 */  
ip2d ip_server2[2] /* this is a 2D array to store 2 IP addresses of Server2 */
```

Some local data objects are also used to store values of message fields.

3.2.3. PROCESSES

In the proposed model, there are four executable processes, two for modeling the behaviors of the two servers, and the other two for the clients.

3.2.3.1. CLIENT PROCESSES

Two client processes are used to model the behaviors of two clients with the same functionality but with different values of channel name and other local variables. The procedure rules for the client process are shown in the flowchart of Figure 3.

Six local variables, namely: *rxid*, *sxid*, *sid1*, *sid2*, *ip_client* and *count* are used in these processes to indicate, respectively, the current values of the received and sent transaction ID, the server IDs of Server1 and Server2, the IP address and the count of the timeouts while waiting for getting an acknowledgement after a request message is sent.

At first, a client broadcasts a 'disc' (discovery) message to both servers, and we have modeled it by sending two messages sequentially using the same (chosen randomly) transaction ID and a server ID and an IP address with zero values. Then, the client waits for collecting offer message(s) and selects one arbitrarily after checking whether the offer is for it or not by matching it with its sent transaction ID. Although, according to the RFC, the client collects a number of offer messages over a period of time, in the proposed model we have chosen just to select one arbitrarily, proceed accordingly, and discarded others. The client timeouts, i.e., breaks its waiting period and retransmits a discovery message if it does not receive offer messages after a specified time.

As shown in Figure 3, the selection of either offer follows the same procedure, and the client uses the server ID of the selected server along with the incremented transaction ID while sending the 'req' (request) message to all the servers. In the flowchart, it is shown for Server1's offer being selected.

The client waits for a positive or a negative acknowledgement message (ack or nak) and makes necessary checks when it receives them. It timeouts and goes back to send a 'request' message for four times and then to the initial state, if it does not get any. All offer messages (in this example, coming from Server2), received in this state are discarded. If a 'nak' message is received then the client goes back to the initial state. Upon the reception of an expected 'ack' message, the client follows a number of options as specified in the RFC (see Section 2.5). We have modeled these options using an if-else selection procedure, and shown them in the flowchart using parallel lines. It sends a 'decline' message if it finds that the offered IP is used by another client, otherwise it remains in a "Bound" state. When the lease time is expired, it goes for a "Renew/ Rebind" state which we have not modeled, but it is shown in the flowchart with dotted lines. The client may go to termination directly or by sending a 'release' message.

Message loss is also modeled by skipping the steps of receptions/transmissions, and enclosed within a #if LOSS-#endif procedure, and defining 'LOSS' as a global variable. It helps in selecting and unselecting the message loss option using the same model.

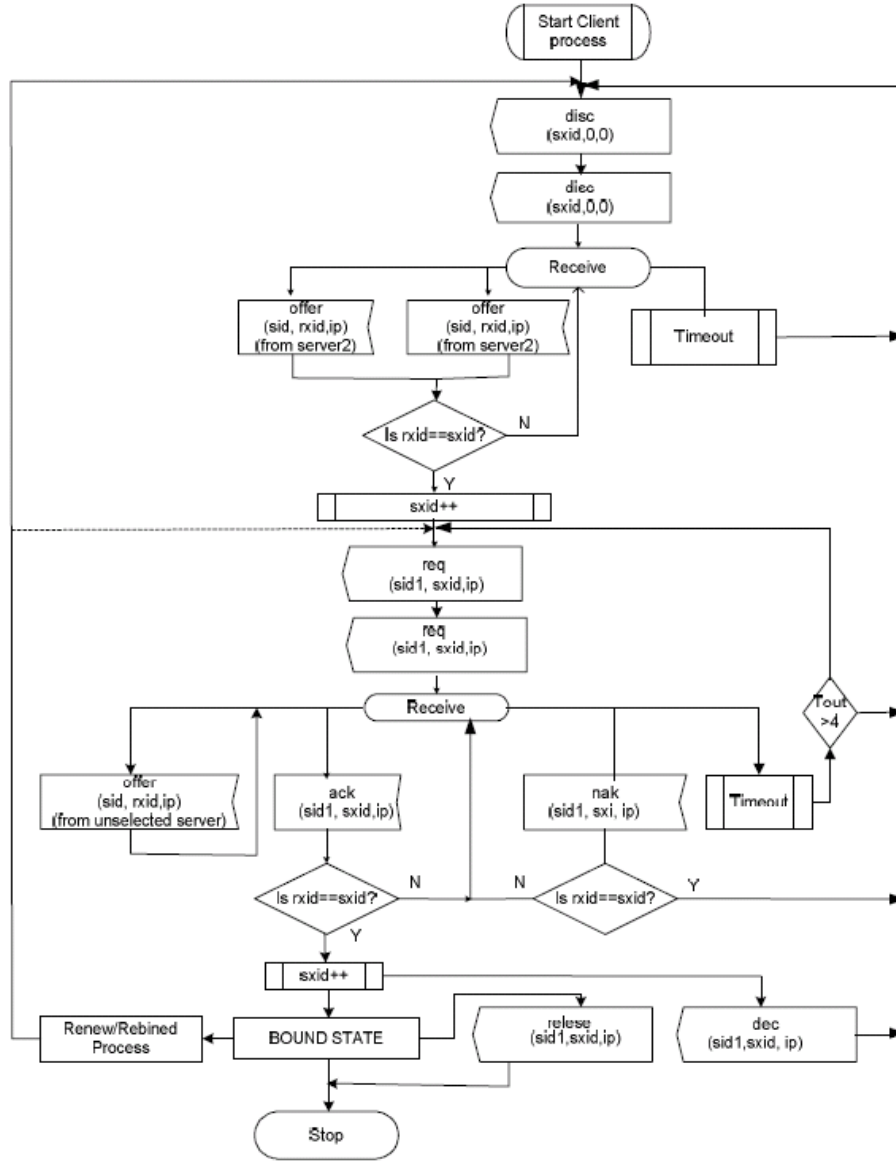


Figure 3: PROMELA procedure of the Client process.

3.2.3.2. SERVER PROCESSES

There are also two server processes for two servers with similar functionality but using different message channels, IP address list and other local variables. The local variable includes *rsid*, *ssid*, *rxid*, *rcvd_ip* and *sent_ip* for storing, respectively, the current values of received and sent server ID, received transaction ID, and received and sent IP addresses. The procedure rules for a server with one client are shown in the flowchart of Figure 4.

In the server process, an infinite loop is defined to receive 'disc', 'dec' or 'release' messages. After getting a 'disc' message, the server checks for the availability of IP addresses and sends an 'offer'

message if any. Then it waits in a loop for getting a 'req' message. It checks the rsid field in the 'req' message and if it does not match with its own, then it marks the IP address available and goes to the initial state. If it matches, then it follows two options: sending a 'nak' if it finds that "IP address" already assigned, or sending an 'ack' message otherwise. Message loss is also modeled in the server processes.

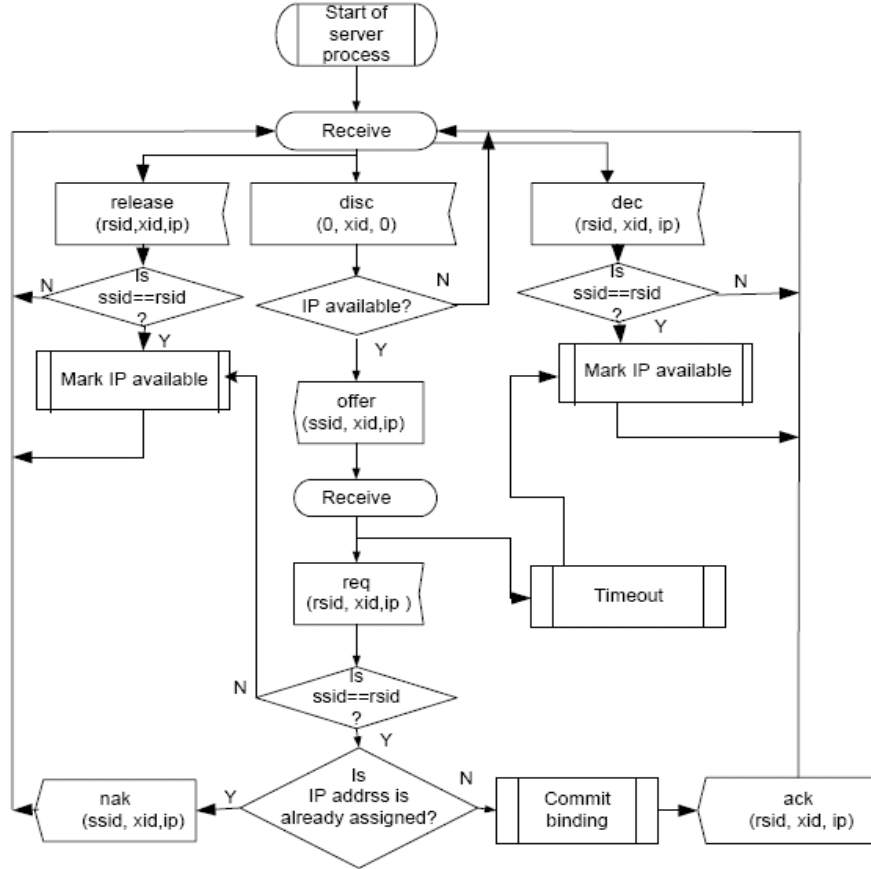


Figure 4: PROMELA procedure of the Server process with a single Client.

In the server process, an infinite loop is defined to receive 'disc', 'dec' or 'release' messages. After getting a 'disc' message, the server checks for the availability of IP addresses and sends an 'offer' message if any. Then, it waits in a loop for getting a 'req' message. It checks the **rsid** field in the 'req' message and if it does not match with its own, then it marks the IP address available and goes to the initial state. If it matches, then it follows two options: sending a 'nak' if it finds that "IP address" already assigned, or sending an 'ack' message otherwise. Message loss is also modeled in the server processes.

4. SIMULATION AND VERIFICATION

The fractions of process behavior that, for one reason or another, are considered suspicious are verified using Spin under various conditions such as message loss or arbitrary errors. Some of the results of the simulation and verification for invalid end-states, non-progress cycles and never claims are discussed in the following sub-sections.

4.1. SIMULATION

One simulation result is shown in the following message sequence diagrams of Figure 5 (the left part shows the beginning of the simulation, while the right part shows the last few steps). In the figures, the two vertical lines from the left are, respectively, for Server1 and Server2 processes, and the next two are, respectively, for Client1 and Client2 processes. Horizontal bars show the respective process execution steps. We obtained the expected execution of the protocol when no message loss ('LOSS') is defined. As shown in the figure, Client1 send a discovery message to both the servers and receives an offer at first form the Server1. So it sends a request message to both servers using a server ID of Server1, and thus informing the rejection of the Server2's offer. It then continues with Server1 for the completion of the message communication. Client2 also gets the first offer message from Server1, but it receives a negative acknowledgement from the server. Then it sends a discovery message again to both the servers and gets the first offer again from Server1. The process is completed when it receives an acknowledgment message from Server1.

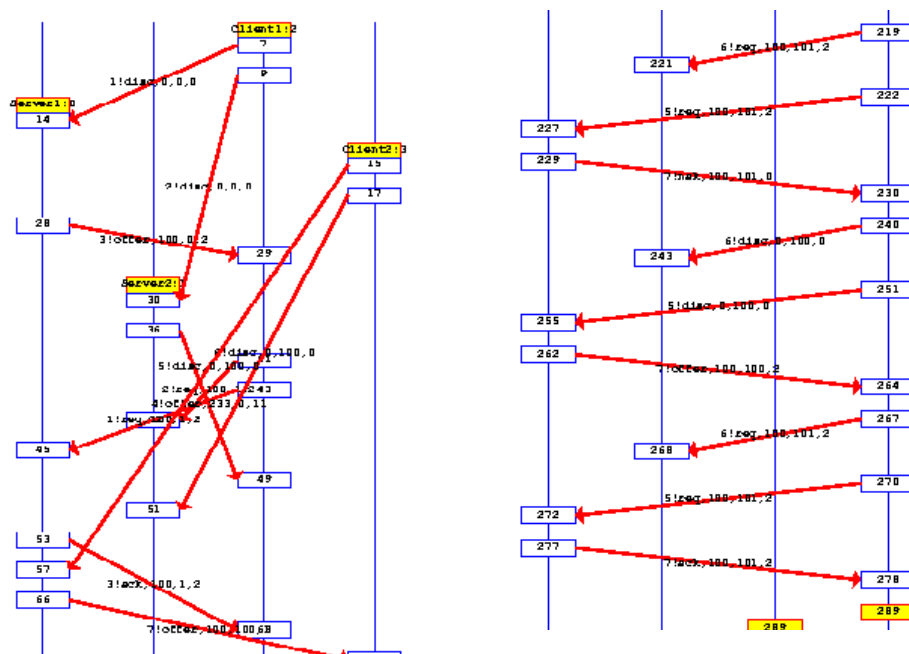


Figure 5: Message sequence diagrams.

4.2. VERIFICATION

4.2.1. INVALID END-STATE

In PROMELA, valid end-states are those system states in which every process instance and the init process have either reached the end of their defining program body or are blocked at a statement that has a label that starts with the prefix ‘end’. Valid end-states also require channels to be empty. All other states are invalid end-states. We found two invalid end-states when there was no client process running. This is because the server(s) process infinitely waits for a ‘disc’ message to be received. These errors are eliminated when we add the ‘end’ labels to Server1 and Server2 processes. With the client process included and ‘LOSS’ not defined, we did not get any invalid end-state. For a similar reason, an invalid end-state was found when a message loss was defined for a ‘disc’ message. No invalid end-state was found as all the other message losses are overcome in the protocol by considering timeouts for waiting for lost messages.

4.2.2. NON-PROGRESS CYCLE

A non progress cycle (NPC) is an execution that does not visit infinitely often a progress state. A progress state is any system state in which some process instance is at a statement having a label that starts with the prefix 'progress'. An NPC was found when there was no server process. This is because after sending the discovery message, the client waits for an offer message; and after a timeout, it restarts the initialization process. This feature is supported in the protocol and therefore the label 'progressive' was added and the error was eliminated. No NPC was found when 'LOSS' is defined.

4.2.3. NEVER CLAIMS

A straightforward verification of 'tasks', or 'requirements', can be modeled as never-claims using PROMELA. We can also formalize a task that is claimed to be performed by the system using Linear Time Temporal logic (LTL) formula. Spin can quickly either prove or disprove that claim [4]. In our model, two LTL claims are used to check the correction criteria. These are discussed below:

- **LTL#1 (!([p-> <> X(<>q))]):** In this LTL, the claim p is defined as reaching the label L2 of the client process when an 'offer' message from the Server2 process is received and q is defined as reaching the label L1 of the Server2 process when a 'req' message is received. Thus, this LTL claims that if an 'offer' message is received by the client, it will eventually send a 'req' message. The operator 'X' is used to avoid the case where q becomes true in precisely the same state as where p becomes true. No violation of the claim was found because a client sends a request message to all the servers from which it got an offer message. The equivalent never claim of this LTL is as follows:

```
#define p (Client@L2)
#define q (Server2@L1)
never { /* LTL: !([p-> <> X(<>q)) ) */
T0_init:
    if
    :: (p) -> goto accept_S5
    :: (1) -> goto T0_init
    fi;
accept_S5:
    if
    :: ! (q) -> goto accept_S5
    fi;
}
```

- **LTL#2 (!([p_ip])):** This LTL is used to verify whether two clients may get the same IP address. We make here the assumption that the two clients get valid IP addresses. Here p_ip is defined as (ip_client1!=ip_client2). Therefore the claim is violated when the same IP address is assigned to both clients concurrently. The equivalent never claim of this LTL is as follows:

```
never { /*LTL: !([p_ip) */
T0_init:
    if
    :: (! ((p_ip))) -> goto accept_all
    :: (1) -> goto T0_init
    fi;
accept_all:
    skip}
```

The protocol ensures that, before offering, a server checks whether the IP to be offered is already given to someone. The same check is done by the client before accepting the acknowledgement. But after running the verification for the above claim, one error is found (see Figure 6). This happened when Server2 offered an IP to Client2 and didn't receive a request on time. After a timeout, it marked that IP as available and gave the same to Client1 getting a discovery message from that. This error can be avoided by increasing the server's timeout value and/or not reusing the offered IP addresses.

```

Full statespace search for:
never claim      +
assertion violations      + (if within scope of claim)
acceptance cycles      - (not selected)
invalid end states - (disabled by never claim)

State-vector 152 byte, depth reached 179, errors: 1
4470 states, stored
855 states, matched
5325 transitions (= stored+matched)
39 atomic steps
hash conflicts: 2 (resolved)
(max size 2^18 states)

```

Figure 6: Verification output of never claim (LTL#2)

5. CONCLUSION

The Dynamic Host Configuration Protocol (DHCP) is studied according to the concept of modeling and verification in this paper. Although we have modeled a specific feature of the protocol, this can be easily extended to model and test other features as well. The same model with few modifications can be used to verify other protocols for the assignment of IP addresses such as Mobile IP. DHCP is a well defined protocol and the proposed model simulation and verification results confirm its claims within the scope of our work. No deadlock or non-progress cycle were found. The work also shows that it is easy to model a communication protocol and the necessary correction claims in PROMELA. SPIN is also found to be very useful and efficient in the protocol verification.

ACKNOWLEDGEMENT

The authors acknowledge and thank King Fahd University of Petroleum and Minerals (KFUPM) for its support and resources to carry out this study.

BIBLIOGRAPHY

- [1] Holzmann, G. J., "The Model Checker SPIN," IEEE Transactions on Software Engineering, Vol. 23, No. 5, May 1997.
- [2] Holzmann, G. J., "The Spin Model Checker: Primer and Reference Manual," Addison-Wesley, September 2003, chapter 3.
- [3] Spin Overview Document, <http://spinroot.com/spin/whatispin.html>
- [4] Spin Manual, Online: <http://spinroot.com/spin/Man/Manual.html>
- [5] Leon-Garcia, A. and Widjaja, I., "Communication Networks: Fundamental Concepts and Key Architectures," International Edition, 2003, McGraw-Hill Companies, Inc., pp. 587-588.

- [6] Tanenbaum, A.S., "Computer Networks", 3rd Edition, 1999, Prentice-Hall, New Delhi, pp. 420-424.
- [7] Droms, R., "Dynamic Host Configuration Protocol," IETF Request for Comments 2131, March 1997. <http://www.ietf.org/rfc/rfc2131.txt?number=2131>.
- [8] Wobus, J "DHCP FAQ".http://www.dhcp-handbook.com/dhcp_faq.html
- [9] Kurose, J. and Ross, K., "Computer Networking: A Top down Approach Featuring the Internet," 2nd edition, Addison-Wesley, July 2002, pp. 329-342.
- [10] Havelund, K., Lowry, M. and Penix, J., "Formal analysis of a space-craft controller using SPIN", IEEE Transactions on Software Engineering, Aug. 2001, Vol. 27 (8), pp. 749 – 765.
- [11] Gluck, P.R. and Holzmann, G.J, "Using SPIN model checking for flight software verification", In Proceedings of IEEE Aerospace Conference, 2002, Vol.1, pp.1-105 - 1-113.
- [12] Khan, S. and Waheed, A., "Modeling and Formal Verification of IMPP," SERP'03, June 23-26, 2003, Las Vegas, Nevada, USA.
- [13] Li Yongjian, and Xue Rui, "Using SPIN to model cryptographic protocols", In Proceedings of International Conference on Information Technology: Coding and Computing, 2004.. ITCC 2004, Vol. 2, 2004, pp. 741 – 745.
- [14] Attiogbe, J.C., "Systematic derivation of a validation model from a rule-oriented model: a system validation case study using PROMELA/SPIN", In Proceedings of International Conference on Information and Communication Technologies: From Theory to Applications, 2004, 19-23 April 2004, pp.581 – 582.
- [15] Xu Gang and Wu Zhiming, "A new method for FMS modeling and formal verification", In Proceedings of IEEE Conference on Emerging Technologies and Factory Automation, 2003.. ETFA '03, 16-19 Sept. 2003, Vol. 1, pp. 224 – 231.
- [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, 2004. SEFM 2004,28-30 Sept. 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, 2005. ECBS '05. 4-7 April 2005, pp. 378 – 385.
- [18] Alur, R. and Wang, B.-Y., "Verifying network protocol implementations by symbolic refinement checking", In Proceedings of the 13th International Conference on Computer-Aided Verification, 2001. <http://www.cis.upenn.edu/~alur/onlinepub.html>.
- [19] Floris, A., Tosetti, and, L. and Veltri, L., " Solutions for mobility support in DHCP-based environments," IEEE International Conference on Communications, 2003, ICC '03, 11-15 May 2003, Vol. 2, pp.1043 – 1047.
- [20] Tominaga, Akihiro, Nakamura Osamu, Teraoka Fumio, Murai Jun, 1995,"Problems and Solutions of Dynamic Host Configuration Protocol (DHCP)", INET'95 Hypermedia In Proceedings, Online: <http://www.isoc.org/HMP/PAPER/127/html/paper.html>
- [21] Droms, R. and Lemon, T., "The DHCP Handbook," Macmillan Technical Publishing, Indianapolis, IN, 1999.