Journal of Theoretical and Applied Information Technology
15th August 2021. Vol.99. No 15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

PROVING THE CORRECTNESS CONDITIONS OF THE THREE-WAY HANDSHAKE PROTOCOL USING COMPUTATIONAL TREE LOGIC

[1]AHMAD ALOMARI, [2] RAFAT ALSHORMAN

[1, 2] Computer Science Department, Faculty of Information Technology & Computer Sciences, Yarmouk University, Irbid-Jordan

E-mail: [1]ahmad.alomari56@yahoo.com, [2]r.alshorman@yu.edu.jo

ABSTRACT

The three-way handshake protocol is widely used especially as a part of complex communication and security systems. It is used to establish a connection between a client and a server under specific rules and constraints. In this research, we used the NuSMV model checker along with Computational Tree Logic (CTL) to verify the correctness of the three-way handshake protocol over specific correctness conditions and properties. The results showed that the proposed protocol satisfied all correctness conditions except δ_9, δ_{11}, and δ_{12}. Furthermore, the proposed automated verification approach aims to verify the correctness of a finite number of clients each of them iterated infinitely often.

Keywords: *CTL, Model Checking, Nusmv, Three-Way Handshake Protocol, Correctness Conditions, Kripke Model.*

1. INTRODUCTION

1.1 Model Checking

Model checking is one of the most important proof techniques used for proving a given correctness condition of a finite-state model. It checks whether the input model satisfies a specific property or not [1]. An automated model checking tool verifies automatically the correctness of a system. It is a verification tool that simplifies the verification of a given system. Thus, it mainly minimizes the efforts spent on the verification task, as well as the time needed for this task [2].

Model checking techniques work by encoding the system that is intended to be verified into an abstract model, and the correctness conditions into temporal logic (CTL and/or LTL). Then, the model checker interprets the abstract model into a kripke model. Thus, the correctness conditions are verified over the kripke model to determine the correctness of the system [2]. If the correctness conditions are satisfied by the system, the model checker will return true [3], [4]. Otherwise, the system will generate a counterexample for each unsatisfied correctness condition to show the errors of the system.

1.2 Temporal Logic

Temporal logic is one of the most important types of logic that is widely used for describing correctness conditions (properties) of different systems. Thus, these properties will be checked automatically using a model checker to determine their correctness. Temporal logic is defined as a type of logic where the truth and falsity varying over time and interpreted over a graph [5]. The motivation behind the temporal logic is that many statements are varying over time (can be true or false at any point in time). However, the propositional logic cannot interpret them. The temporal logic is divided into two models; the Computation Tree Logic (CTL), and the Linear Time Logic (LTL). LTL is a type of temporal logic where a sequence of states is interpreted linearly over time [5]. The CTL is a type of logic where the time is branched in a form of a tree structure and called branching time logic [3].

Temporal logic is one of the most important types of logic because it provides the ability to branch time over specific tree structures [5]. Moreover, it can be used for describing the behaviors of concurrent complex systems [6], [7].

1.3. The Proposed Three-Way Handshake Protocol

The three-way handshake protocol is a process used to establish a connection between client and a server. In our proposed work, we intend to automate the proving process of the three-way handshake protocol. Therefore, we will try to verify the correctness of this protocol over specific correctness conditions using an automated model checker (e.g., NuSMV).

The benefits of the three-way handshake protocol can be illustrated into two main points; it can be used to establish a connection between communicated parties, and it is used as a security process to protect the network from malicious activates. For example, the TRAP is a three-way handshake protocol used for detecting the flooding of the synchronize packets and the Distributed Denial of Service (DDoS) attacks [17].

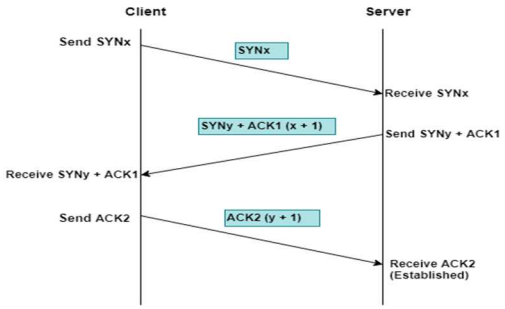

Figure 1. The three-way connection establishing process.

Figure *1*, represents how the three-way handshake protocol operates to establish a connection. For example, assume that there are one client and one server that wants to communicate with each other using this protocol. The client starts the process by sending a Synchronize packet (SYNx) to the server. x represents the sequence number of the client which is a random synchronization number generated by the client and sent to the server to establish a connection. When the server receives the client SYN x, it replies by a Synchronize-Acknowledge packet (SYNy-ACK1). y represents a random synchronization number generated by the server and sent to the client. ACK1 represents the sequence number of the client x plus one. SYNy and ACK1 sent to the client to inform it that the server has received SYNx. ACK1 is considered as the first acknowledgment. When the client receives the server SYN-ACK1, it replies by ACK2 packet that contains the sequence number of the server y plus one. ACK2 represents the second acknowledgment. When the server receives this packet, it can establish a connection with the client or reject it based on specific rules.

Our proposed verification approach differs from the previous verification techniques in three ways as follows:

1. First, model checking techniques can conduct proofs for infinitely often processes. Therefore, model checking can verify and interpret all of the three-way handshake protocol behaviors, and it is based on rigorous proof. Unlike other mathematical proving techniques that are not based on rigorous proof [10], [11].

2. Second, model checkers generate counterexamples if the protocol contains errors, unlike other mathematical proving techniques where this is not possible. The generated counterexamples will help in the enhancement process of the protocol by showing its errors (unsatisfied properties).

3. Third, model checking techniques increases the effectiveness of the proving process by reducing the errors that could arise from conducting the proof using other mathematical proving techniques [12], [13], [14].

2. RELATED WORK

Zbrzezny, B and Kurkowski in [18] used the Satisfiability Module Theories based on Bounded Model Checking (SMT-BMC) techniques along with synchronized timed automata networks, to verify the correctness of the Needham-Schroeder Public key (NSPK), the Needham-Schroeder Shared Key (NSSK), and Wide-Mouthed Frog (WMF) security protocols using this verification approach. In [19], the authors used a continuous verification method based on the Cryptographic Protocol Shapes Analyzer (CPSA) to verify the correctness of the Pair-wise Shared Key Establishment (PSKE) and Group Shared Key Establishment (GSKE) security protocols. Another study in [20] aimed at using the COQ model checker to verify the correctness of *the* NSPK, and the NSSK Protocols over specific safety properties and attack models. In [21] Cremers *et al* used the tamarin model checker to validate the TLS version 1.3 security protocol. In [22], the authors used the induction method along with the Isabelle/HOL theorem prover to verify the

correctness of the Franklin-Reiter sealed-bid auction protocol and the NSSK protocol. Another study proposed in [23] aimed at providing a verification approach for verifying the correctness of the TLS handshake protocol based on the miTLS protocol. Fu and Koné in [24] used The Input-Output Labeled Transition System (IOLT) to represent the NSSK protocol to be verified using the SG-IOLT model. Alshorman in [4] used the NuSMV model checker to verify the correctness of the TCP protocol. In [25], the author developed a vehicular communication system and verified it using the Casper/FDR formal verification tool. Another study in [26] aimed at providing an open-source model checker that is called MCMAS for verifying the correctness of multi transactions systems such as the SPORE protocol and the anonymity protocols.

Alshorman in [27] verified the correctness of the debits and credits transactions using the NuSMV over specific CTL and LTL serializability properties. In [28] Alshorman and Fawareh developed an efficient verification approach based on conflict graph reduction for verifying the correctness of a specific multi-transaction system. Alshorman and Hussak in [29] verified the correctness of a scheduler that schedules the infinite incoming and outcoming multi-transactions of a system using the NuSMV over specific LTL serializability properties. In [30], Alshorman and Hussak verified the correctness of multi-transactions systems for accessing uniform data using the NuSMV over specific CTL and LTL serializability properties.

Most of the efforts conducted for proving the correctness of security and communication protocols limits the number of clients and servers. They used formal verification methods or model checking based on simulation [18, 20, 21, 24, 26]. The drawback of these techniques is that all the possible behaviors of a protocol cannot be verified. Moreover, the authors may use pure mathematical or theorem proving techniques [19, 22, 23, 25]. The problem with mathematical proofs is that it is difficult to write equations to represent abstract models of systems. Also, theorem proving cannot verify the correctness of a system that changes it is states over time. This is because they are based on propositional logic that can verify the systems that do not change over time.

3. THE PROPOSED AUTOMATED VERIFICATION APPROACH

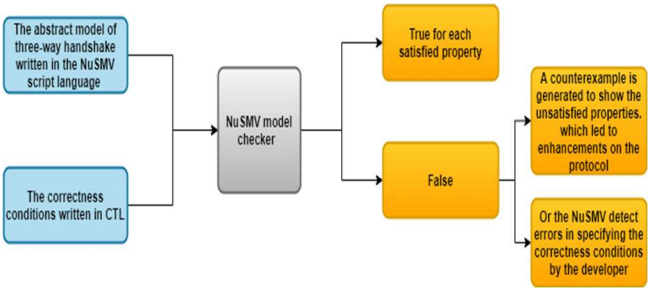

Figure 2. The workflow for the automated model checking approach.

In figure 2, the flowchart of the proposed automated verification approach is illustrated to specify how the verification approach will verify the correctness of the three-way handshake protocol. Thus, the automated verification process consists of n steps as follows:

1. The three-way handshake protocol will be encoded into an abstract model checking algorithm using a specific model checker script (e.g., NuSMV script).

2. The proposed correctness conditions (properties) that we assumed for verifying the correctness of the proposed protocol, will be encoded into temporal logic formulas (CTL).

3. The model checker will interpret the abstract model into kripke structure, to be verified over the proposed properties.

If the three-way handshake protocol satisfied all the correctness conditions, the model checker will return true for each property. Otherwise, the model checker will return false as a counterexample to show the unsatisfied properties which may lead to enhancement on the protocol. If there are any errors in specifying the properties in temporal logic, the model checker will return a compilation error.

4. KRIPKE STRUCTURE AND TEMPORAL LOGIC

4.1. Kripke Structure

Kripke structure is a type of directed graph that consists of nodes and edges, where nodes represent the reachable states of a specific system

(model), and edges represent the transitions among the states. Each state is labeled with specific properties, and each path in the graph describes a specific behavior of the modeled system [8]. Kripke structure is crucial for temporal logic because it is used to provide temporal logic (CTL and LTL) with definitions when a specific property is satisfied [9].

Model checking techniques suffer from the state space explosion problem which indicates that there is an exponential increase in the state space, making it cannot be handled by the capacity of the memory [15]. Therefore, we intend to generate the kripke structure that represents the three-way handshake protocol using the NuSMV model checker because it avoids this problem.

4.2. CTL

In this study, we have used CTL to represent the proposed correctness conditions because it is more expressive than LTL for the proposed model. Moreover, this model contains many branching times, therefore CTL can describe it correctly.

The CTL is a temporal logic type where the time is branched and interpreted over a specific branching-time graph (e.g., Kripke structure). Thus, each state in a model that is represented using CTL has more than one successor [31].

A model that is specified using CTL is interpreted over a specific branching-time structure, such that there exist path quantifiers for verifying the CTL model over specific paths. The syntax of the CTL consists of the set of atomic propositions (e.g., p, q), ordinary boolean operators $(\neg, \wedge, \vee, \top, \bot)$, propositional quantifiers (A, E), and four temporal logic operators **X**, **F**, **G**, **U**. Therefore, any statement in CTL can be represented by:

$\Phi ::= \mathrm{pr_i} \mid \neg\varphi \mid \varphi 1 \vee \varphi 2 \mid \varphi 1 \wedge \varphi 2 \mid \mathrm{AX}\varphi \mid \mathrm{E}[\varphi 1 \cup \varphi 2] \mid \mathrm{EX}\varphi \mid \mathrm{AF}\varphi \mid \mathrm{EF}\varphi \mid \mathrm{AG}\varphi \mid \mathrm{EG}\varphi \mid \mathrm{A}[\varphi 1 \cup \varphi 2]$

where $\mathrm{pr}_1, \mathrm{pr}_2, \ldots, \mathrm{pr}_i$ identifies any atomic proposition used in the three-way handshake protocol [4].

The CTL semantics can be derived from the fact that the CTL is interpreted over branching-time structures. Thus, a branching time model $M = (S, \rightarrow, L)$ where S represents the set of the model states, $\rightarrow$ is the transition relation, and L is the function that is used to label each state in the model with specific atomic propositions [4].

Let $M = (S, \rightarrow, L)$ be a transition model such that $s \in S$, $\varphi \in P$ where s represents a state in M (belongs to S), P represents the set of all possible equations (all the possible correctness conditions that are written in CTL), the truth values; false and true represented by $\perp$ and $\top$ symbols respectively. Thus, ordinary operators' semantics are as follows: s_i

1. $M\ s_i \vDash \neg \phi$ iff $M, s_i \nvDash \phi$.
2. $M\ s_i \vDash \phi \wedge \psi$ iff $M,\ s_i \vDash \phi$ *and* $M, s_i \vDash \psi$.
3. $M, s_i\ \phi \vee \psi\ iff\ M, s_i = \phi$ or $M, s_i\ \psi$.
4. $M, s_i\ \phi \rightarrow \psi\ iff\ M, s_i \vDash \phi$ then $M, s_i \vDash \psi\ M, s_i \vDash \top\ M, s_i \nvDash \perp$.

Let $\lambda = (s_i, s_{i+1} \ldots)$ such that λ represent an outgoing path from a specific state s_i in M. Thus, the semantics of the CTL operators are as follows:

1. $M, s_i \vDash \mathrm{AX}\,\varphi\ iff,\ \forall\lambda = (s_i, s_{i+1} \ldots),\ M, s_{i+1} \vDash \varphi$.
2. $M, s_i \vDash \mathrm{EX}\ \varphi\ iff,\ \exists\lambda = (s_i, s_{i+1} \ldots),\ M, s_{i+1} \vDash \varphi$.
3. $M, s_i \vDash AF\,\varphi\ \ iff,\ \ \forall\lambda = (s_i, s_{i+1} \ldots),\ \exists\, j \geq i,\ M, s_j \vDash \varphi$.
4. $M, s_i \vDash \mathrm{EF}\,\varphi\ iff, \exists\lambda = (s_i, s_{i+1} \ldots), \exists\, j \geq i,\ M, s_j \vDash \varphi$.
5. $M, s_i \vDash AG\ \phi\ iff,\ \forall\lambda = (s_i, s_{i+1} \ldots)$, and $\forall j, j \geq i, M, s_j \vDash \phi$.
6. $M, s_i \vDash EG\ \phi\ iff,\ \exists\lambda = (s_i, s_{i+1} \ldots)$, and $\forall j, j \geq i, M, s_j \vDash \phi$.
7. $M, s_i \vDash A[\phi_1\ U\ \phi_2]\ iff, \forall\lambda = (s_i, s_{i+1} \ldots),\ \exists\ j \geq i$ such that $M,\ s_j \vDash \phi_2$, *and* $\forall k, i \leq k < j, M, s_k \vDash \phi_1$.
8. $M, s_i \vDash E[\phi_1\ U\ \phi_2]\ iff, \exists\lambda = (s_i, s_{i+1} \ldots)$ such that, $\exists\, j \geq i\ \ M,\ s_j \vDash \phi_2$, *and* $\forall k, i \leq k < j, M, s_k \vDash \phi_1$.

According to the above semantics, we can ensure that there is branching in the CTL future, as there are many paths to go.

In this study, CTL is used to encode the correctness conditions and the properties of the three-way handshake protocol. Therefore, the

NuSMV model checker is used to verify whether the proposed three-way handshake protocol satisfies these conditions and properties or not. If the proposed protocol satisfied all these properties, The NuSMV will generate true for each satisfied property. In case there is unsatisfied property, the NuSMV will generate a counterexample to show the set of states, in the model, where this unrequired property is violated.

CTL and LTL have incomparable expressive power. In LTL the time is linear; which means that each possible execution is represented as a linear line of a sequence of states [32]. On the other hand, in CTL there is branching in time; which means that there is more than one execution path. This means that many properties can be expressed using only LTL, and many properties can be expressed using only CTL. Thus, the choice between CTL and LTL depends on the system and the correctness conditions you intend to verify.

For example, the LTL formula $GFp \rightarrow GFq$ is only satisfied using LTL, not CTL. Therefore, in all paths, this formula will eventually be satisfied by the model where p represents a statement (Baier and Katoen, 2008). $GFp \rightarrow GFq$ is not equivalent to $EG(AFp) \rightarrow AG(AFq)$ $(GFp \rightarrow GFq \not\equiv EG(AFp) \rightarrow AG(AFq))$. $EG(AFp) \rightarrow AG(AFq)$ represents the CTL formula of the LTL formula $GFp \rightarrow GFq$. However, CTL is not suitable to be used to verify this type of formulas, and this example is explained in details in section 7.

5. THE SYNCHRONOUS MODEL OF ITERATED CONCURRENT CLIENTS AND SERVER

In this part of the proposed study, we assume that there is a finite number of clients each of them requests the server infinitely often to establish a connection. We used the NuSMV model checker to generate the kripke structure of this model.

The kripke model of the iterated concurrent clients and server model is shown in figure 4. The connection between a concurrent client and the server is synchronous. This means that the client and the server will synchronously send their states to each other, which provides reliable communication between them. Therefore, the server will not move to the next state until the client takes action, and the client also will not move to the next state until the server takes action.

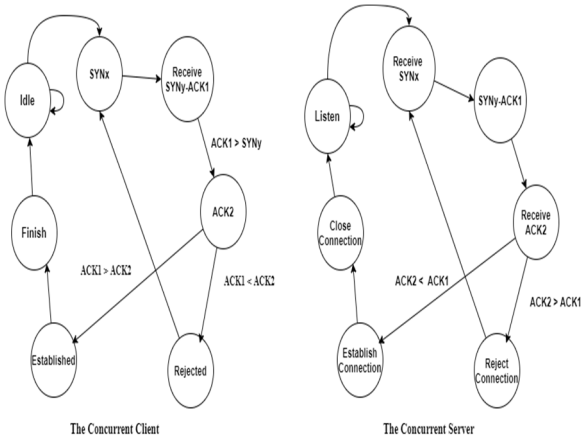

Figure 3. The kripke model of the three-way handshake protocol.

Figure 3 shows the finite state model of the iterated concurrent clients and server communicating using the three-way handshake protocol. In this finite state model x, represents the sequence number of the client (a random synchronization number sent by the client to the server), and y represents the sequence number of the server (a random synchronization number sent by the server to the client). ACK1 represents the sequence number of the client x plus 1, and it is an acknowledgment sent by the server. ACK2 represents the sequence number of the server y plus 1, and it is an acknowledgment sent by the client.

In our proposed iterated model, we assume that x is greater than y. Therefore, to establish a successful connection between a concurrent client and the server, ACK1 has to be greater than ACK2. The transitions among the proposed finite state model are as follows:

1. The client starts the execution of the protocol by moving from the Idle state to the SYNx state, which indicates that the client has sent a request that contains x to the server to establish a connection.

2. When the server receives this request, it moves from the Listen state to the Receive SYNx state. Then, the server moves to the SYNy-ACK1 states, which in this state it sends y and ACK1 to the client indicating that the server has received the client request.

3. When the client receives y and ACK1 from the server, it moves to the Receive SYNy-ACK1 if and only if ACK1 is greater than y. Then, the client moves to the ACK2 state, which in this state it sends

ACK2 to the server indicating that the client has received y and ACK1.

4. When the server receives ACK2, it moves from the previous state to the Receive ACK2 state. Therefore, if $ACK2 < ACK1$, the server moves from the previous state to the establish connection state. Otherwise, the server moves to the Reject connection state. If $ACK1 > ACK2$ the client moves from the previous state to the Established state indicating that the connection is established by the server. Otherwise, the client moves to the Rejected state indicating that the connection is rejected by the server.

5. When the server satisfies the client request, it moves from the previous state to the Close Connection state; indicating that the connection is terminated by the server. On the other hand, the client moves from the previous state to the Finish state when the server terminates the connection.

6. After the server successfully ends the connection and the client request has been satisfied, the client goes back to the Idle state to request the server infinitely often. On the other hand, the server goes back to the Listen state to satisfies more client's requests.

7. A client can stay idle forever if it not requesting the server, and the server can also stay listening forever if there are no requesting clients.

The verification process of the proposed iterated concurrent clients and server model is not trivial. So, we need to efficiently prove this model by verifying all of its possible behaviors. Thus, we used CTL to encode and describe the correctness conditions of this model. However, it is critical to automate the proving process of a finite number of clients iterated infinitely often using model checkers to reduce the errors that arise from conducting traditional proofs.

6. THE PROPOSED CORRECTNESS CONDITIONS

In this proposed work, we intend to prove the correctness of the three-way handshake protocol by verifying general correctness conditions for synchronous processes and specific correctness conditions for the protocol model. Most protocols proofs assume that the number of requests is finite. Thus, in this research, we assume that the client's requests are iterated infinitely often. Therefore, the proposed protocol is not correct until it satisfies the following correctness conditions (encoded into CTL):

1. As the three-way handshake protocol is a three-way connection establishing mechanism, then at least there will be one satisfied request SYN from a client c_i to the server, S to establish a successful connection between them.

$$\delta_1 = \bigwedge_{1 \le i \le n} AG((c_i = SYN) \rightarrow AF(c_i = Established)) \quad (1).$$

n represents the finite number of clients, and i represents the number of the current client, SYN represents the request of the client c_i, and $Established$ represents the state where the connection is established between the client and the server.

2. Each client c_i must request the server infinitely often (many times).

$$\delta_2 = \bigwedge_{1 \le i \le n} (AG((c_i = SYN) \rightarrow AF(c_i = Established)) \wedge AG((c_i = Establish) \rightarrow AF(c_i = SYN))) \quad (2).$$

If a client c_i has successfully established a connection with the server S, then after c_i finish sending data to S and S terminates the connection, c_i can request the server infinitely often (iterated infinitely often).

3. The server S must satisfy each client c_i infinitely often requests.

$$\delta_3 = \bigwedge_{1 \le i \le n} (AG((S = Receive\_SYN) \rightarrow AF(S = Establish\_Connection)) \wedge AG((S = Establish\_Connection) \rightarrow AF(S = Receive\_SYN))) \quad (3).$$

4. At least there will be one client c_i which is currently in the Idle state and eventually will finish sending the data to the server successfully (before the termination of the connection by the server).

$$\delta_4 = \bigwedge_{1 \le i \le n} AG((c_i = Idel) \rightarrow AF(c_i = Finsih)) \quad (4).$$

$Finsih$ represents the state where c_i finish sending data to the server.

5. We must have two acknowledgments such that the first acknowledgment $ACK1$ precedes the second acknowledgment $ACK2$. $ACK1$ lies at the beginning of the communication process, and $ACK2$ lies at the end of this process. When a client c_i receives $ACK2$, this indicates that the connection is established (or may be rejected by the server S).

$$\delta_5 = \bigwedge_{1\le i\le n} AG(((S = Send\_SYN\_ACK1) \wedge (c_i = ACK2)) \rightarrow AX(S = Receive\_ACK2)) \quad (5).$$

$Send\_SYN\_ACK1$ represents the state where S sends $ACK1$, and $ACK2$ represents the state where c_i has received $ACK1$ and sends $ACK2$ to S. $Receive\_ACK2$ represents the state where S has received $ACK2$ from c_i.

6. A client c_i request SYN must precede $ACK1$ and $ACK2$; $SYN < ACK1 < ACK2$, where $<$ means occurs before.

$$\delta_6 = \bigwedge_{1\le i\le n} (AG((c_i = SYN) \wedge (S = Send\_SYN\_ACK1)) \rightarrow AF(c_i = Receive\_SYN\_ACK1) \wedge (S = Receive\_ACK2) \quad (6).$$

7. $ACK2$must precede the data sending (before establishing the connection); $ACK2 < Establish\_Connection$.

$$\delta_7 = \bigwedge_{1\le i\le n} AG((S = Receive\_ACK2) \rightarrow AF(S = Establish\_Connection)) \quad (7).$$

8. If we have two clients; client c_i and client c_j such that c_i requests SYN_i and c_j requests SYN_j the server S, then, request SYN_i should precede request SYN_j; $SYN_i < SYN_j$ where $i \neq j$.

$$\delta_8 = \bigwedge_{1\le i\le n, 1\le j\le n, i\neq j} AG(((c_i = SYN_i) \rightarrow EF(c_j = SYN_j)) \rightarrow ((c_i = Established) \rightarrow AF(c_j = Established))) \quad (8).$$

9. If a client c_i is currently in the $Idle$ state, c_i can stay in the $Idle$ state forever.

$$\delta_9 = \bigwedge_{1\le i\le n} EG(AF(c_i = Idle)) \rightarrow AG(AF(c_i = Idle)) \ (CTL).$$

$$\delta_9 = \bigwedge_{1\le i\le n} G(F(c_i = Idle)) \rightarrow G(F(c_i = Idle)) \ (LTL).$$

This correctness condition is encoded in both CTL and LTL. Therefore, based on the verification results for each of these CTL and LTL formulas, we will be able to see the difference between CTL and LTL in describing this correctness condition.

10. If a client c_i is rejected by the server S, the client can request the server infinitely often many times.

$$\delta_{10} = \bigwedge_{1\le i\le n} AG((c_i = Rejected) \rightarrow AF(c_i = SYN)) \ (10.CTL).$$

$Rejected$ represents the state where the request of the client c_i is rejected by S.

11. If a client c_i has sent $ACK2$ to the server S, and S is currently in the $Receive\_ACK2$ state, then, always the next state of c_i is the $Rejected$ state.

$$\delta_{11} = \bigwedge_{1\le i\le n} AG(((c_i = ACK2) \wedge (S = Receive\_ACK2)) \rightarrow AX(c_i = Rejected)) \quad (11).$$

12. If a client c_i is currently in the $ACK2$ state, then, always the next state of c_i is the $Established$ state.

$$\delta_{12} = \bigwedge_{1\le i\le n} AG((c_i = ACK2) \rightarrow AX(c_i = Established)) \quad (12).$$

Let $\emptyset = \bigwedge_{1\le i\le 12} \delta_i$,where $\emptyset$ represents the conjunction of all the correctness conditions $(\delta_1, \delta_2, \dots, \delta_{12})$, and δ_i is the current correctness condition. Now, we will build an abstract model M of the proposed three-way handshake protocol, such that M represents the kripke structure of the proposed protocol. Therefore, given M, and $\emptyset \in$ CTL, M is considered true and correct iff $M \vDash \emptyset$ in all states. This means that all the correctness

conditions $M_i \models \delta_i$ represented by a model M in all states.

7. RESEARCH RESULTS

This section consists of two subsections. The first subsection describes the keywords and the variables of the proposed NuSMV model; how they are encoded into the NuSMV script to represent the three-way handshake protocol. In the second subsection, the crucial part of the proposed correctness conditions results is shown.

7.1. The Proposed NuSMV Model Script and Used Variables

In this part of the proposed study, we have encoded the three-way handshake protocol into an abstract model using the NuSMV script to be able to check automatically if the abstract model satisfies the proposed correctness conditions. NuSMV script is a low-level coding language used for representing finite state systems.

We have encoded the proposed protocol into a NuSMV abstract mode using several keywords and variables as follows:

- MODULE: Keyword that identifies the main module and submodules.
- VAR: Keyword that declares variables.
- SPEC: Keyword that defines the proposed CTL and LTL correctness conditions.
- ASSIGN: Keyword used to identify the transition relations among the variables.
- init: Keyword used to define the initial values of the variables.
- next: Keyword used to declare a relationship between the variables in a specific state or and its inheritor state (defines the next values of the variables).
- MODULE Client (y, A1, SS): declares the model of the client such that; y represents the sequence number of the server, A1 represents the first acknowledgment, and SS represents the set of the server states. y, A1, and SS will be *synchronously* sent to the client by the server.
- c_state: Variable that represents the set of the client states.
- c_seq: Variable that represents the sequence number of the client.
- MODULE Server (x, A2, CS): declares the model of the server such that; x represents the sequence number of the client, A2 represents the second acknowledgment, and CS represents the set of the client states. x, A2, and CS will be *synchronously* sent to the server by the client.
- s_state: Variable that represents the set of the server states.
- s_seq: Variable that represents the sequence number of the server.
- MODULE main: Identifies the main model from where the code will start executing.
- s: Variable of type MODULE Server defines the server that will provide clients with the connection.
- c1: Variable of type MODULE Client defines the first client.
- c2: Variable of type MODULE Client defines the second client.

7.2. The Proposed Correctness Conditions Results

When we have checked the proposed correctness conditions in the NuSMV, the results showed that some of these correctness conditions ($\delta_1, \delta_2, \dots, \delta_8$, and δ_{10}) are satisfied and some are not ($\delta_9, \delta_{11}, \delta_{12}$) explained in counterexamples. Therefore, the proposed correctness conditions ($\delta_1, \delta_2, \dots, \delta_{12}$) that are written in CTL were unfolded in NuSMV script and checked. The following figures are part of the results using the NuSMV:

1. SPEC AG ((s.s_state = Receive_SYN) -> AF (s.s_state = Establish_Connection)) & AG ((s.s_state = Establish_Connection) -> AF (s.s_state = Receive_SYN)).
2. A. SPEC AG ((c1.c_state = Idle) -> AF (c1.c_state = Finish)).

B. SPEC AG ((c2.c_state = Idle) -> AF (c2.c_state = Finish)).

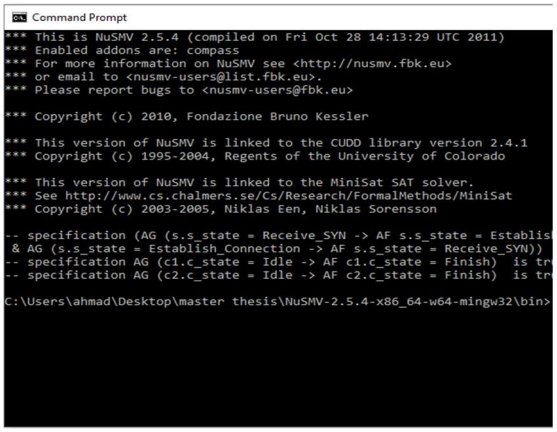

Figure 4. The results of the correctness conditions δ_3, δ_4

In figure 4, the results of the correctness conditions δ_3, δ_4 are true in all of the model M states. Thus, these correctness conditions are satisfied by M.

1. A. SPEC EG (AF (c1.c_state = Idle))-> AG (AF (c1.c_state = Idle)).

B. LTLSPEC G (F (c1.c_state = Idle)-> G (F (c1.c_state = Idle))).

C. SPEC EG (AF (c2.c_state = Idle))-> AG (AF (c2.c_state = Idle)).

D. LTLSPEC G (F (c2.c_state = Idle)-> G (F (c2.c_state = Idle))).

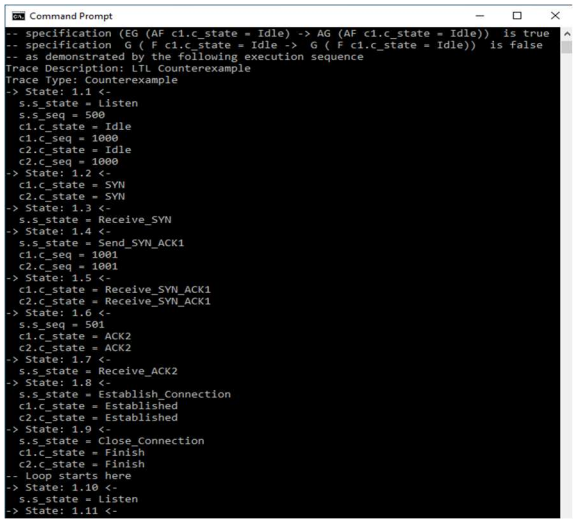

Figure 5. The results of the correctness δ_9.

The correctness condition δ_9 indicates a linear path. Therefore, we have used LTL to express this correctness condition and verify it correctly. This correctness condition cannot be expressed using CTL because CTL is interpreted on many paths of the proposed model. This means that using the quantifier E in CTL will include an incorrect path resulting in incorrect verification. As shown in figure 5, the verification result of δ_9 using LTL is false. Therefore, the NuSMV generated a counterexample to show the set of states where δ_9 is not satisfied by the model M.

The verification results of δ_9 proves that LTL and CTL have incomparable expressive power by ensuring that many properties can be expressed using only CTL, and others using only LTL.

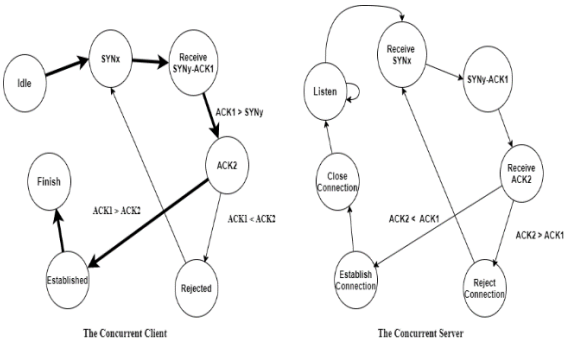

Figure 6. The modified kripke model of the three-way handshake protocol.

As shown in figure 6, we have modified the abstract model of the three-way handshake protocol to obtain the correct verification results of δ_9. We have removed the self-loop of the idle state and the path between the finish state and the idle state. The blooded path in figure 11 indicts that a client c_i will eventually be in the finish state forever, and no path leads to the idle state. This path proves the correctness of the verification results that are generated by the NuSMV for δ_9 in figure 6. Therefore, our modification on the proposed model proves it is correctness without been modified.

8. CONCLUSION

The main aim of this study is to propose and use the CTL to verify the correctness of the synchronous processes that are competing to share resources based on specific protocol rules. The three-way handshake protocol one of the most widely used protocols in networking, security, and mobile computing because this protocol can establish a reliable and efficient connection between

concurrent iterated clients and a specific server. In this study, we verified the correctness of the three-way handshake protocol concerning specific properties using the NuSMV.

We have proved the correctness of a finite number of concurrent clients each of them requests the server infinitely many times by automating the proving process using the NuSMV. Also, we have proposed a kripke structure that represents the synchronous model of the iterated concurrent clients and the server. Moreover, we have discovered that CTL is very efficient and suitable for encoding synchronous correctness conditions. Furthermore, the proposed automated verification approach increases the effectiveness of the proving process by reducing the errors that could arise from conducting the proof using other mathematical proving techniques such as simulation.

The results showed that the correctness conditions $\delta_1, \delta_2, \ldots, \delta_{10}$ were satisfied by the proposed protocol, which means that these correctness conditions are true in all of the protocol states. On the other hand, the correctness conditions δ_{11}, δ_9, and δ_{12} were not satisfied by the protocol, and counterexamples were generated by the NuSMV to show the states where these unsatisfied correctness conditions are violated. Moreover, the correctness conditions δ_2, δ_3 proves that the proposed model of the three-way handshake protocol is a finite number of clients iterated infinitely often.

For future work, we intend to prove the correctness of complex security protocols that the three-way handshake protocol is a part of these protocols, and this is called prove by construction.

REFERENCES

[1] Baier, C. and Katoen, J. (2008) *Principles of Model Checking*. Cambridge, Mass: The MIT Press Ltd.

[2] Baier, C. and Katoen, J. (2008) *Principles of Model Checking*. Cambridge, Mass: The MIT Press Ltd.

[3] Alshorman, R. and Fawareh, H. (2013) 'Reducing conflict graph of multi-step transactions accessing ordered data with gaps', *Annals of the University of Craiova-Mathematics and Computer Science Series*, 40(1), pp. 1–8.

[4] Alshorman, R. (2019) 'Toward Proving the Correctness of TCP Protocol Using CTL', *The International Arab Journal of Information Technology*, 16(3), pp. 407–414.

[5] Venema, Y. (2017) 'Temporal logic', in Goble, L. (ed.) *The Blackwell Guide to Philosophical Logic*. first edit. new jersey: Blackwell, pp. 203–223.

[6] König. H. (2012) *Protocol Engineering*. Berlin: Springer-Verlag Berlin Heidelberg.

[7] Konur, S. (2013) 'A survey on temporal logics for specifying and verifying real-time systems', *Frontiers of Computer Science*, 7(3), pp. 370–403.

[8] Molnár, V. *et al.* (2016) 'Component-wise incremental LTL model', *Formal Aspects of Computing*, 23(3), pp. 345–379.

[9] Torres, P. J. R. *et al.* (2018) 'Probabilistic Boolean network modeling and model checking as an approach for DFMEA for manufacturing systems', *Journal of Intelligent Manufacturing*. Springer US, 29(6), pp. 1393–1413.

[10] Nickel, G. (2019) 'Aspects of freedom in mathematical proof', *ZDM*. Springer Berlin Heidelberg, 51(5), pp. 845–856.

[11] Jongsma, C. (2019) 'Mathematical Induction and Arithmetic', in *Introduction to Discrete Mathematics via Logic and Proof*. 1st edn. Cham: Springer, pp. 149–204.

[12] Antonini, S. (2019) 'Intuitive acceptance of proof by contradiction', *ZDM*. Springer Berlin Heidelberg, 51(5), pp. 793–806.

[13] Kaur, K. and Kumar, S. (2013) 'Analysis of various testing techniques', *International Journal of System Assurance Engineering and Management*, 5(3), pp. pages276–290.

[14] Boci, I. and Nicolás, B. (2019) 'Inductive verification of data model invariants in web applications using first-order logic', *Automated Software Engineering*, 26(2), pp. 379–416.

[15] Aceituna, D. and Do, H. (2019) 'Addressing the state explosion problem when visualizing off - nominal behaviors in a set of reactive requirements', *Requirements Engineering*. Springer London, 24(2), pp. 161–180.

[16] Hsu, F. *et al.* (2016) 'applied sciences TRAP : A Three-Way Handshake Server for TCP Connection Establishment', *Applied Sciences*, 6(11), p. 358.

[17] [17] Boro, D. and Bhattacharyya, D. K. (2017) 'DyProSD : a dynamic protocol specific defense for high - rate DDoS flooding attacks', *Microsystem Technologies*. Springer Berlin Heidelberg, 23(3), pp. 593–611.

[18] Zbrzezny, A.M., Szymoniak, S. and Kurkowski, M. (2020) 'Efficient Verification

of Security Protocols Time Properties Using SMT Solvers', in Martínez Álvarez, F. et al. (eds) *International Joint Conference: 12th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2019) and 10th International Conference on EUropean Transnational Education (ICEUTE 2019). CISIS 2019, ICEUTE 2019. Adva*. Cham: Springer, pp. 25–35.

[19] Molina-markham, A. and Rowe, P. D. (2017) 'Continuous Verification for Cryptographic Protocol Development', in *SafeThings'17 Proceedings of the 1st ACM Workshop on the Internet of Safe Things*. Delft: ACM, pp. 51–56.

[20] Ligatti, J. (2017) 'POSTER : Towards Precise and Automated Verification of Security Protocols in Coq', in *CCS '17 Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas: ACM, pp. 2567–2569.

[21] Cremers, C. *et al.* (2016) 'Automated Analysis and Verification of TLS 1 . 3 : 0-RTT , Resumption and Delayed Authentication', in *2016 IEEE Symposium on Security and Privacy (SP)*. San Jose: IEEE, pp. 470–485.

[22] Everson, J., Lawrence, M. and Paulson, C. (2015) 'Verifying multicast-based security protocols using the inductive method', *International Journal of Information Security*, 14(2), pp. 187–204.

[23] Bhargavan, K. *et al.* (2014) 'Proving the TLS Handshake Secure (As It Is)', in Garay J, A. and Gennaro, R. (eds) *Advances in Cryptology – CRYPTO 2014. CRYPTO 2014. Lecture Notes in Computer Science*. Berlin: Springer, pp. 235–255.

[24] Fu, Y. and Koné, O. (2013) 'A Finite Transition Model for Security Protocol Verification', in *SIN '13 Proceedings of the 6th International Conference on Security of Information and Networks*. Aksaray: ACM, pp. 368–371.

[25] Bae, W. S. (2015) 'Function-based connection protocol development and verification', *Cluster Computing*. Springer US, 18(2), pp. 761–769.

[26] Vinh, T. and Levente, T. (2013) 'On automating the verification of secure ad-hoc network routing protocols', *Telecommunication Systems*, 52(4), pp. 2611–2635.

[27] Lomuscio, A., Qu, H. and Raimondi, F. (2017) 'MCMAS : an open-source model checker for the verification of multi-agent systems', *International Journal on Software Tools for Technology Transfer*. Springer Berlin Heidelberg, 19(1), pp. 9–30.

[28] Alshorman, R. (2016) 'Temporal Logics Specifications for Debit and Credit Transactions', *International Journal of Information Technology and Computer Science*, 7(5), pp. 10–17.

[29] Alshorman, R. and Fawareh, H. (2013) 'Reducing conflict graph of multi-step transactions accessing ordered data with gaps', *Annals of the University of Craiova-Mathematics and Computer Science Series*, 40(1), pp. 1–8.

[30] Alshorman, R. and Hussak, W. (2010) 'specifying a Timestamp-based Protocol For Multi-step Transactions Using LTL', *International Journal of Computer and Information Engineering*, 4(11), pp. 1716–1723.

[31] Alshorman, R. and Hussak, W. (2009) 'A CTL Specification of Serializability for Transactions Accessing Uniform Data', *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 3(3), pp. 780–786.

[32] Doczkal, C. and Smolka, G. (2016) 'Completeness and Decidability Results for CTL in Constructive Type Theory', *Journal of Automated Reasoning*. Springer Netherlands, 56(3), pp. 343–365.