Modelling and Verification of CoAP over Routing Layer using SPIN Model Checker

Anchal J Vattakunnel, Suresh Kumar N∗, G Santhosh Kumar

Department of Computer Science, Cochin University of Science and Technology, Cochin-682022, India

Abstract

Many of the communication protocols developed for the resource constrained devices are rarely subjected to protocol verification. Design flaws like deadlocks, livelocks, non-progressive cycles etc. may come into view during the realization and can cause catastrophic effects in safety-critical applications. Formal specification of the protocol represented as a model helps to describe and analyse the conformability of the implementation to its specification and subsequently reveal design flaws, if any in the system. The formal model is subject to verification by inserting correctness and safety properties of the protocol and validating logical correctness using a model checking tool. All model checkers suffer from state explosion problem due to enormous states of the model being created. The key contribution of the present work is the introduction of a method to develop a compact verification model which is amenable to full state space search by abstracting the key elements of a protocol. Moreover, many protocol verification works presented in the existing literature consider only a single layer of a communication protocol. To correctly model the overall behaviour of a protocol, interactions between the layers have to be incorporated. The proposed method has been proven useful by considering verification of an application protocol, CoAP for constrained devices by abstracting out the aspects of the underlying routing protocol RPL. Reliable message exchanges among various entities are modeled and its safety and correctness properties were analysed and verified. The results obtained show that the model performs the full state space search by considering all possible routing paths and are free from design flaws. The method described has been implemented by building a validation model in PROMELA and the model is verified by using SPIN model checker. The methodology used in this paper can be used to verify any application layer protocol for constrained devices in IoT scenario that run on top of routing layer.

Keywords: COAP; SPIN; PROMELA; Model checking; Model Verification; IoT

1. Introduction

Internet of Things (IoT) consists of millions of IP connected smart devices interacting and communicating with each other over the web. The exponential increase in the sensor assisted applications put new demands on continuous sharing of information at global scale. It is expected that by 2020 there will be 50 billion of connected things1 and

∗ Corresponding author.
E-mail address: anchaljacob@gmail.com, sureshkumar@cusat.ac.in, san@cusat.ac.in
communication protocols play a major role. The development of communication protocols are usually start from the RFC documents where the specifications given in these are mapped into implementation directly using a suitable programming language. Such protocols deployed on devices may result in failure during later stages of execution due to the design flaws. These flaws can only be identified and addressed by performing formal verification during the development. Formal verification of protocols include building models from its specifications followed by inserting properties that are to be verified into the model in the form of temporal claims, assertions and monitors. This is then fed to a model checker that performs an exhaustive search of all possible sequence of execution paths that matches with the negation of the specification properties. For any execution path that matches with the negated property specification, model checker generates a violating trace showing execution path that leads to property violation. Every execution path generated by the model checker is maintained as a sequence of execution states in system memory. The number of states generated by the model checker grows exponentially with respect to the size of the model. This may lead to state explosion problem resulting memory overflow. Hence, the size of the model should be minimized without losing core properties of the protocol. The constrained devices used in IoT applications make use of Routing Protocol for Low Power and Lossy Networks (RPL)\(^2\) proposed by IETF at the routing layer for multi hop communication. RPL protocol constructs a direct acyclic graph (DAG) based on some objective function. Formal verification of application layer protocol for constrained devices requires interaction with routing layer. Modeling RPL for verification of application layer protocol leads to increase in the size of the model which results in state explosion.

This paper proposes a method to abstract out routing characteristics for the verification of application layer protocols for constrained devices in IoT. The overall outline of the proposed concept is depicted in Figure 1. By using this proposed method, verification of an application layer protocol namely Constrained Application Protocol (CoAP) for constrained devices is attempted. Protocol is modeled using PROMELA\(^3\) and is verified by SPIN\(^5\) model checker.

The paper is organized as follows. Section II describes related works. Proposed design methodology for formal verification of CoAP is presented in Section III. The design elements of CoAP are discussed in Section IV. The Modeling and formal verification of CoAP incorporating proposed routing strategy is presented in Section V. Conclusion and future works are discussed in section VI.

2. Related Works

Various software tools, methods and proposals designed to improve quality of routing protocols using formal verification techniques are presented in\(^6\). The work also present various types of formal verification techniques, the
problem of state explosion and its solution, an algorithm for formal verification and its illustration by modeling the various protocols using PROMELA. Various techniques involved in the verification of correctness properties for communication protocols based on finite state machines are depicted in7,8. The work also suggest different types of design flaws that are identified and addressed by formal verification. A systematic framework used for choosing modeling languages for cyber physical systems is presented in9. The framework describes the selection of modeling tool based on stakeholders view, type of formalism used. It also suggests language tools like SPIN, UPPAAL10 and NuSMV for the verification of models based on finite state machine and hybrid automaton.

The key concepts involved in model checking, description of SPIN model checker, tools used for demonstrating concurrency, nondeterminism are described in11. Classification of model checkers consisting of explicit state model checkers, bounded model checkers, constraint satisfaction model checkers and symbolic model checkers are presented in12. It also describes various temporal languages like LTL,CTL etc for property specification. The various ways in which the design flaws of a protocol can be identified and addressed are explained using a simple protocol for distributed systems presented in13. An attempt to compare the performance of bitstate hashing method with other hashing techniques for efficient state space search is presented in14. A method to derive UIO sequence from PROMELA specification for the solution of conformance test problem was presented in15. A method to integrate timing elapsed between events was suggested in16. The effectiveness of the above method was analysed and compared by incorporating timing features in PAR and BRP protocols.

An algorithm for building an abstract model for protocols related to mobile ad-hoc networks was described in17. The algorithm described was validated for LAR, DREAM and OLSR ad-hoc routing protocols. A technique used for the verification of Wireless Adaptive Routing Protocol(W.A.R.P) using SPIN was described in18. Modeling and verification of DVR protocol using SPIN was depicted in19. Verification of DHCP protocol starting from abstracting of design elements and its behavior was described and safety properties were analyzed using SPIN in20. The messaging frame work for EAP protocol was modeled and the basic safety properties of the protocol were analyzed in21. Collision avoidance protocol was modeled and verified using SPIN and UPPAAL model checkers in22 and its safety properties were analyzed. FTSP (Flooding Time Synchronization Protocol) was modeled using PROMELA and verified using SPIN in23. The Safety properties pertaining to time synchronization of FTSP were verified and analyzed using LTL.

The Evolution and standardization of various protocols with a view to support IoT were described in24. It also propose a standardized protocol stack for constrained devices using IEEE 802.15.425 PHY layer and MAC layer, 6LOWPAN26, RPL/ROLL27 for network layer, UDP for transport layer and CoAP28 for application layer. An overview about CoAP and its standardization by CoRE (Constrained RESTful Environment) is discussed in29. This work describes the interoperability between a constrained environment with the traditional internet environment using HTTP and CoAP protocol stacks. The performance of CoAP and HTTP in terms of energy consumption, response time and bytes transferred per transaction was analyzed in30.

The literature review shows that only few works were focusing on verification of IoT protocols. Many works related with protocol verification was done by considering message exchanges within the same layer. Modeling the protocol behavior for cross layer communication is vital for IoT protocols. Hence a verification method for application layer protocol (CoAP) by abstracting its interaction with routing layer for constrained devices communicating through multiple hops is proposed. This paper focuses mainly on a method to abstract out routing layer for the verification of CoAP.

3. Proposed design Methodology

Multi hop communications are inevitable for smart devices due to their limited range. It is required to incorporate the characteristics of multi hop routing for performing formal verification of application layer protocols. In this paper routing concepts are embedded into the application layer protocol by constructing all the possible routing paths that exist among nodes. A separate verification covering all possible paths has been performed for destination nodes containing multiple paths. The proposed derivation of routing information and verification of CoAP protocol are based on the extracting the design elements of protocols as suggested in4. Consider the topology described in Figure 2. For the verification of CoAP node 0 is considered as source node. The possible paths between source (nodes 0) and the node with the highest level (node 8) are as shown below. Path 1 ->0-1-5-8 (3 hops), Path 2 ->0-1-5-7-8 (4 hops),
path 3 ->0-3-5-8 (3 hops), path 4 ->0-3-5-7-8 (4 hops) and Path 5 ->0-4-6-7-8 (4 hops). Separate routing information are constructed for each path. Here five routing matrices are required to store the routing information along the five different paths. The routing information is maintained as a MxM matrix for topology consisting of M nodes. The algorithm for updating routing matrices are depicted in Algorithm 1. Applying algorithm on the multi hop topology

Algorithm 1 Routing Table Construction Algorithm
1. For each destination node j, 1 <= j <= n, compute the path P_j = r_1, r_2, ..., r_k that exist from source node 0 to j.
2. For each path in set P_j repeat the following steps:
   2.1. Let r_1 be a path consisting of q_{11}, q_{12}, q_{13} ..., q_l be the set of nodes contained in one of the path of destination node j.
   2.1.1. Set Routing matrix[q_{11}, q_{12}] = q_{12}, and Routing matrix[q_{12}, q_{11}] = q_{11}.
   ... Set Routing matrix[q_{k-1}, q_k] = q_k, and Set Routing matrix[q_k, q_{k-1}] = q_{k-1}.
3. If there exists one path in P_j update all the routing matrices.
4. If there are 'm' multiple paths then update 'm' routing matrices with each element in P_j along 'm' paths.

shown in Figure 2 result in the path obtained from each destination nodes and are as follows.

P1=[0 ⇔ 1], P2=[0 ⇔ 2], P3=[0 ⇔ 3], P4=[0 ⇔ 4], P5=[q1 = 0 ⇔ 1 ⇔ 5, q2 = 0 ⇔ 3 ⇔ 5], P6=[0 ⇔ 4 ⇔ 6], P7=[q1 = 0 ⇔ 1 ⇔ 5 ⇔ 7, q2 = 0 ⇔ 3 ⇔ 5 ⇔ 7, q3 = 0 ⇔ 4 ⇔ 6 ⇔ 7], P8=[q1 = 0 ⇔ 1 ⇔ 5 ⇔ 8, q2 = 0 ⇔ 3 ⇔ 5 ⇔ 8, q3 = 0 ⇔ 4 ⇔ 6 ⇔ 8, q4 = 0 ⇔ 1 ⇔ 5 ⇔ 7 ⇔ 8, q5 = 0 ⇔ 3 ⇔ 5 ⇔ 7 ⇔ 8]

The routing paths constructed as per Algorithm 1 by considering two paths from node at lowest level (node 0) to highest level (node 8) are shown in Table 1-2. The routing information for the other three paths from node 0 to node 8 can be constructed in the same way.

4. Design Elements of CoAP

The design elements of CoAP are evolved from RFC 7252 and are based on the concepts described in 4.

a) SERVICE Specification

It is a protocol used for web transfer among constrained devices and connected via internet. CoAP implements a RESTful environment for constrained devices. RESTful nature of this protocol is realized by using Request/Response
architecture. CoAP assumes every smart objects used in IoT as a list of URIs. Here a client request a particular service from a RESTful server by specifying the URI. The services include sensing and actuation. The server responds to the client via replay message. A particular server can handle several requests and services. The protocol should provide reliable service that can withstand link failures.

b) ASSUMPTION about environment
i). The components involved are CoAP Clients that request the service to the CoAP server by specifying its URI followed by service specification. The request can be cached through intermediate nodes that act as proxy.
ii). The transmission channels are full duplex, wireless and lossy.
iii). Transport layer protocol used is UDP.
iv). Network and MAC layer used are 6LOWPAN and IEEE 802.15.4.

c) VOCABULARY
CoAP supports the following messages.
i). GET : Used to retrieve the resource specified by the URI-path
ii). PUT : Used to Update or actuate specific bits / devices identified by URI-path.
iii). DELETE : Used to remove a specific service from the device identified by URI-path.
iv). POST : Used to create a specific URI path for later use.
v). ACK : It is the message issued from the server side in response to a confirmable message.
vi). RST: If the message is unable to process by the server it will responds with a RST message indication that message can’t be processed.

d) FORMAT
Format of CoAP frame is shown in Figure 3 Version (Ver): 2-bit unsigned integer, the CoAP version number it is set to 1 (01 binary).
Type (T): 2-bit unsigned integer indicating message types.
Code: 8-bit unsigned integer indicating the type of message request and response.

e) PROCEDURE RULES
1. A CoAP client can send any one of the following message types which may be either a confirmable (CON) or Non-Confirmable (NON).
   1.1 GET, POST, PUT & DELETE. Client will retransmit the same message if it doesn't receive the ACK within the ACK timeout interval for CON message. Client will also perform retransmission if it doesn't receive any response for NON messages.
   1.2 Number of retransmission from the client side will be limited to maximum retransmission limit specified by the protocol.
2. When a server receives a confirmable message described in rule 1.1 it can respond either by sending a piggybacked response with ACK, response code, payload and message-ID or as a separate response with ACK, response code and message-id without payload.
   2.1 In case of separate response server initially sends ACK with response code 0.00 indicating that the response will be sent at later time. When the server is free it will send a CON message with payload and code 2.05 and its token specified in CON request received earlier.
   2.1.1 If the server do not receive a client response in lieu of the ACK message with code 0.00 send in rule 2.1 within a stipulated time the server retransmits the same message.
3. When a server receives a NON message it responds with a response code followed by payload if response is positive.
4. When a client receives a separate response for a CON message, the client sends an ACK with response code 0.00.

5. Modelling and Formal verification of CoAP

5.1. Validation Model for CoAP

A validation model for CoAP was built by using PROMELA. The model abstract out the procedural rules with emphasis given to the CON message sequence associated with GET. This is due to similarity of external behavior that exists among other confirmable messages. The model follows the topology described in Figure 2. The routing behavior was incorporated in accordance with the routing path construction algorithm described in section III. The external message sequence interaction among various nodes of the model generated by using SPIN model checker is depicted in Figure 4 Which describes the message sequence interaction

Here a CON GET message is sent from node 0 to node 8 and its corresponding piggybacked ACK from node 8 to node 0. The process id (pid) assigned by the model checker for node 0, node 1, node 5, node 7 and node 8 involved in message exchanges are 1, 2, 6, 8 and 9 respectively. The path chosen by the model in accordance with Table 2.

5.2. Formal Verification of CoAP

The formal verification was done for a set of nine nodes as per the topology described in Figure 2. Here node 0 was considered as client node and all other nodes treated as server nodes. The properties are inserted in the validation
model and are verified for message interaction between client node and other server nodes separately. The verification was done using bit state hashing and maximum search depth was limited up to 252. The properties that are verified in the model have full state space coverage. Various properties inserted in the model are described below.

Property 1: When a CON GET message is send to any of the destination node it will eventually reach the server node.

The LTL corresponding to this property is described as follows.

\[
\text{ltl p1 ([]}<>S-[]<>C)\]

\(S\) is a boolean variable that will be set to true in the model whenever a message is received from client. Another boolean variable \(C\) is set to true in the model whenever a client sent a message to any destination node. The equivalent never claim for the LTL is given in Listing 1

\(([]<>S-][<>C)\) is

Listing 1. Never Claim for Property 1

```
never { /* ( [])<> (S) -> []<> (C) */
T0_init:
  do :: ((C)) -> goto accept_S10
  :: (1) -> goto T0_S10
  :: (! ((S))) -> goto accept_S26
  :: (1) -> goto T0_S23
od;
accept_S10:
  do :: (1) -> goto T0_S10
od;
accept_S26:
  do :: (! ((S))) -> goto accept_S26
od;
T0_S10:
  do :: ((C)) -> goto accept_S10
  :: (1) -> goto T0_S10
od;
T0_S23:
  do :: (! ((S))) -> goto accept_S26
```
The results obtained after verifying this property using Table 2 was described in Table 3. The results suggest that the

model checker performs full state space searched and the property is satisfied. The verification was repeated using all the remaining routing paths and results show that this property is satisfied.

Property 2: Absence of Non-Progressive cycle

The never claim used for checking non-progress cycles is shown in Listing 2.

Listing 2. Never Claim for Property 2

```c
never { /* []<>np_; */
T0_init:

  do
    :: ((np_)) -> goto accept_S9
  :: (1) -> goto T0_init
  od;

accept_S9:

  do
    :: (1) -> goto T0_init
  od;
}
```

SPIN model checker maintains a reserved in variable named 'np_'. The value of the variable will be set to true by the model checker if there are any execution sequence that cycles through unmarked states. Any occurrence of non-progress cycle causes never claim to cycle through state marked as 'accept_S4'. This will violate the property of absence of non-progress cycles. The verification results showing the absence of non-progress cycle for a CON GET message is as shown in the Table 4.

Table 4. Verification Result: Property 2

<table>
<thead>
<tr>
<th>Destination</th>
<th>Depth Searched</th>
<th>Hash Factor</th>
<th>State Stored</th>
<th>Memory Usage (MB) for States</th>
<th>Pan:Rate States/ Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>192</td>
<td>1.24276e+06</td>
<td>108</td>
<td>127</td>
<td>2.308</td>
</tr>
<tr>
<td>2</td>
<td>192</td>
<td>1.24276e+06</td>
<td>108</td>
<td>127</td>
<td>2.308</td>
</tr>
<tr>
<td>3</td>
<td>192</td>
<td>1.24276e+06</td>
<td>108</td>
<td>127</td>
<td>2.308</td>
</tr>
<tr>
<td>4</td>
<td>192</td>
<td>1.24276e+06</td>
<td>108</td>
<td>127</td>
<td>2.308</td>
</tr>
<tr>
<td>5</td>
<td>212</td>
<td>1.04858e+06</td>
<td>128</td>
<td>153</td>
<td>2.735</td>
</tr>
<tr>
<td>6</td>
<td>212</td>
<td>1.04858e+06</td>
<td>128</td>
<td>153</td>
<td>2.735</td>
</tr>
<tr>
<td>7</td>
<td>232</td>
<td>906877</td>
<td>148</td>
<td>179</td>
<td>3.162</td>
</tr>
<tr>
<td>8</td>
<td>252</td>
<td>798915</td>
<td>168</td>
<td>205</td>
<td>3.59</td>
</tr>
</tbody>
</table>

Property: 3: Absence of deadlock
This can be verified by running the PROMELA model without inserting any claims in the analyzer mode. The model checker flags error if any deadlock occurs. The verification result shows that the presence of zero errors and hence absence of deadlock.

5.3. Results and discussion

The verification was carried after exploring full state space search for all the properties. The maximum search depth reached was 252 and was attained in the communication scenario between nodes belonging to lowest and highest level in the topology described in Figure 2. Specification of CoAP limits the number of communication for facilitating energy efficiency. Hence the number of state transitions occurred in the verification was limited. For the verification of application layer protocols that require continuous message exchanges, the number of state transitions will grow exponentially.

6. Conclusion and Future works

A verification model for application layer protocols in IoT considering its interaction with routing layer was proposed. The proposed concept was illustrated by conducting formal verification of CoAP running on top of a compact form of RPL. A validation model for CoAP was built using PROMELA for a multi hop topology. The external behavior of the model was analyzed by generating a message sequence chart depicting the message flow between nodes. This model was verified by using SPIN model checker covering safety, liveness and correctness properties. The verification results were analyzed in terms of memory usage, number of state transitions and maximum search depth attained for all the inserted properties.

The proposed routing path construction can be applied for the verification of any application layer protocol for constrained devices. The basic correctness properties described for CoAP can be extended to include other message types by incorporating sufficient CoAP protocol abstractions. The routing path construction can also be extended to support dynamic topology changes.

References