2015 IEEE International Conference on Smart City/SocialCom/SustainCom together with DataCom 2015 and SC2 2015

A Study on the Reliability of Software Defined Wireless Sensor Network

Yulin Lu, Xin Huang, Baichuan Huang, Weiwen Xu, Qian Zhang, Ruiyang Xu, Dawei Liu
Department of Computer Science and Software Engineering
Xi'an Jiaotong-Liverpool University
Suzhou, China
Email: Xin.Huang@xjtlu.edu.cn

Abstract—**Software defined wireless sensor network is an network defined for dynamic and secure control of smart devices. It decouples the data plane and the control plane, allowing administrators to reprogram the smart devices in the network and backbone network devices based on users' varying demands. In this paper, a typical architecture of software defined wireless sensor network is proposed. Continuous time Markov chain and continuous stochastic logic are used for model checking the reliability of this architecture, which lead to several interesting findings.**

I. Introduction

Wireless sensor networks (WSNs) provide a new interface between humans and the physical world. They can help people to monitor, track, and control the environment. WSNs normally includes many sensor nodes. These nodes can be imaged as tiny computers: they contain a wireless communications device, an embedded processor, and a power source. They collect and relay sensor data to various applications.

The growing popularity of sensor-based applications places an increasing demand for dynamic and secure services from WSNs. Software-defined wireless sensor network (SDWSN) is a potential solution to address such challenges. First of all, the network becomes programmable by software devices without changing the hardware devices in lower layers. This could isolate each application from others and realize dynamic control over the virtual network. Secondly, the status of network devices could be visible to network administrators, which makes it more flexible to manage the network and add or improve security schemes. Thus, this new networking paradigm is considered as a promising technology.

However, before SDWSN can be widely deployed, its reliability must be taken into consideration. First of all, the problem with the centralized architecture of SDWSN is that the whole network may collapse if the central controller fails. In other words, the SDWSN controller is liable to a single point of failure. Additionally, node failures and communication failures are also important issues that influence the reliability.

In this paper, the SDWSN reliability is studied using model checking techniques. So far, little research has been undertaken using these techniques. The overall aim is to study the influence factors, and find out potential improvement strategies. The potential questions include:

- How can we formally model a SDWSN?
- Whether the following factors influence the SDWSN reliability: the number of central controllers, the failure rate of controllers, the number of sensors, etc.
- If we can find practical strategies (i.e. adjusting these factors mentioned above in a practically acceptable range) to improve the SDWSN reliability?

Model checking SDWSN using probabilistic model involves the following aspects: (i) Find out SDWSN failure types, and develop a formal model of SDWSN using continuous-time Markov chains (CTMC); (ii) Implement the model using the PRISM language, a simple, state-based language; (iii) Verify properties (e.g. if certain factor influence the SDWSN reliability) of the model using continuous stochastic logic (CSL) and costs/rewards.

The main contributions of the work is that we show how SDWSN can be formally modelled using CTMC. Also, we give several interesting findings such as two controller with middle reliability can be better than the combination of one controller with high reliability and one controller with low reliability, sensors are usually the main reason of SDWSN failure, and sensor failure usually happens before other failures.

The remainder of the paper is organized as follows. Section II explains software defined networking. Section III introduces probabilistic model checking techniques. Section IV shows how we formally model a typical SDWSN. Section V analyzes the model checking results. Section VI provide related works. Finally, the conclusion is made in Section VI.

II. Software Defined Networking

Software defined networking (SDN) is a new promising networking paradigm [6], [9]. It provides a way to programmatically control networks by decoupling the network control and forwarding functions. The main components of the SDN (shown in Figure 1) is listed below.

- SDN application: SDN applications are programs designed for network management. Requests such as monitoring and changing the network topology can be sent from these programs. Some of these applications can be built in controllers as plugins.
- SDN control plane: it includes SDN controllers that offer a centralized view of the data forwarding plane, and those manage the routers and switches according to application requirements. A controller of the latter

function, for example, may tackle a request of data transmission by picking a certain switch and passing the transmission instruction to it. It may also add, change or delete the data flows under different conditions.

- SDN data forwarding plane: it is the network that transmits data for applications, following the rules managed by the control plane.
- SDN southbound interfaces: they are interfaces between SDN controllers and the SDN data forwarding plane. Typical protocols like OpenFlow exposes the flow tables to the control plane and thus realizes the centralized network management.
- SDN northbound interfaces: they are interfaces between SDN applications and SDN controllers. For example, OpenDaylight and FloodLight provide the Rest API to enable the communication between SDN applications and controller plugins.

The SDN provides a flexible way to control the network, which is ideal for today s applications with dynamic requirements. Some typical use cases are as follows. SDNs can be used for data centers and cloud, because it offers better resource utilization. It can also be used for enterprise, carrier and service provider, since additional bandwidth or policy can be easily added. Furthermore, the controller has a central view of the whole network, thus SDN architecture can enhance network security.

III. Probabilistic Model Checking

Probabilistic model checking is a formal verification technique. Formal verification refers to the technique of verifying whether a given model meets specific requirements such as safety and robustness. The model as well as the requirements are formulated into mathematical languages. These languages are then passed as inputs to a model checking algorithm.

A. Continuous-Time Markov Chain

SDWSN will be modelled using continuous-time Markov chains (CTMC). A CTMC [5] is a tuple $\mathcal{C} = (S, \bar{s}, R, L)$ where:

- S is a finite set of states;
- $\bar{s}$ is the initial state;
- $R : S \times S$ is the transition rate matrix (a rate is assigned to each pair of states);
- L is a labelling function.

One example CTMC is shown in Fig. 2. s_0, s_1 and s_2 are the states. s_0 is the initial state. Empty and full are labels. The transition rate matrix is

$$\begin{pmatrix} 0 & 1 & 0 \\ 2 & 1 & 0 \\ 0 & 2 & 0 \end{pmatrix}$$

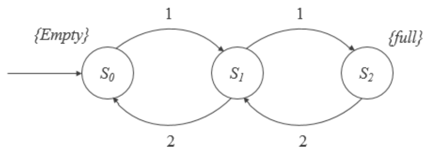

Figure 2. A example of CTMC.

B. PRISM

PRISM [5] is a free and open-source probabilistic model checker. It can be used to model and analyze systems using discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs), probabilistic automata (PAs) and probabilistic timed automata (PTAs). Prism has been deployed in a broad spectrum of application domains such as security protocols, wireless communication protocols, biological systems, randomized distributed algorithms, game theories, planning and synthesis, and many other areas.

Probabilistic models are specified by the Prism language, a set of simple and state-based clauses. Module is the building block. Each module mimics the states and behaviors of a component of the actual systems. For example, in the SDWSN, applications, controllers and network devices shall be viewed as primitive modules. Additionally, each module has a set of states denoted as variables and could perform a set of behaviors denoted as commands. A command is a description about what the module would do in a certain situation with a certain probability. For a CTMC, a command could be depicted as follows:

```
[] <guard> -> <rate> : <action>;
```

For example, consider the following Prism clause:

```
[] (x = 0)-> 0.5 : (x′ = x + 1)
```

This statement claims that x has a probability of 0.5 to increase itself by one when x has a value 0. Thus, it could be intuitively deduced that guard is a predicate over all variables that defined in the module whereas action is a behavior that describes that the transition of states of the module by changing the variable values. The rates are always positive real numbers.

C. CSL, Costs and Rewards

We write specification of CTMCs using the logic CSL (Continuous Stochastic Logic) [5]. For example,

```
P=? [ ! "down" U<=T "fail_cont" ]
```

means that what is the probability of the event: there is no failure until a controller failure happens in time T.

PRISM could analyze systems using 'Costs And Rewards' [5]. It is real values relating to certain state or command within a module. Consider the following example: The piece of Prism statements assigns a reward of 100 whenever x has a value of 4 or 10 and assigns $3 \times x$ as a reward to x wherever the value of x is between 5 and 20.

```
rewards
x = 4 : 3;
```

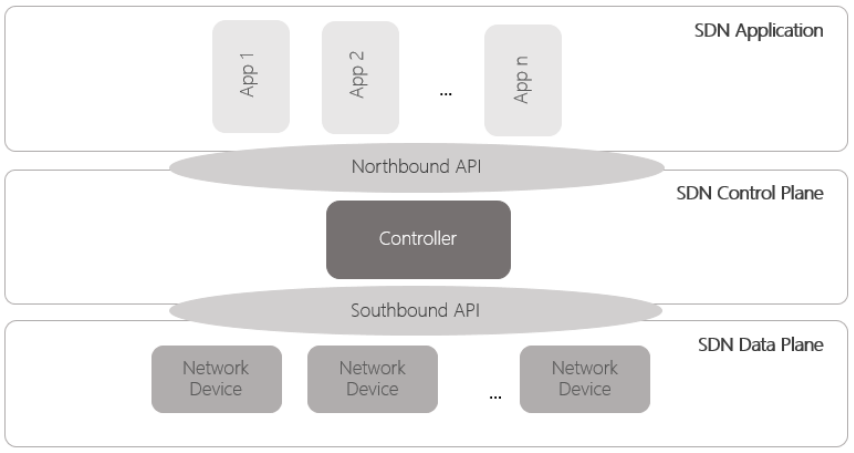

Figure 1. SDN architecture.

```
    x = 10 : 3;
x > 5 & x < 20 : 3 * x;
endrewards
```

In addition, a state may satisfy several preconditions described by guard clause. Then the reward assigned to this state would be the sum of rewards corresponding to each guard clause.

IV. Model Checking SDWSN

This section describes a typical SDWSN architecture, and presents a formal model for it. Also, implementation details using PRISM are introduced.

A. Formal Model of a typical SDWSN

A typical SDWSN architecture is shown in Figure 3. The data plane includes a network device layer and a host device layer. While the routers in the upper layer focus on packet forwarding, the sensors in the lower layer establish a network among themselves. Note that since we do not consider the failure rate of either southbound or northbound interfaces, they are not illustrated in Figure 3.

In a formal model of the architecture, applications send requests to the controller cluster containing two control devices, which processes them and instructs routers to forward packets.

- Control devices: Each of the two controllers has a certain failure rate. In SDWSN, the controller failure refers to the condition when both of the two controllers fail.
- Network devices: In our model, there are three routers in the network device layer. Each of them has a certain failure rate. When one router fails, all the sub nodes of it will be moved to a waiting sequence. After that happens, those in the waiting sequence will be chosen as substitutions of the dead sensors connected to other working routers. The network device layer is considered to be down only when all routers are down. It is worth to be mentioned that the failure rate of a router is correlated to the number of sensors connected to it. This comes from the correlation between the number of connections and the burden of the routers.
- Sensors: Sensor failures can be caused by two reasons. For one thing, the sensor may fail by itself. The failure rate is once per 7 days in this model. For the other, when no router is working properly, which indicates the failure of the whole network device layer, the sensor layer is deemed to fail as well, for there no longer exist usable connections between the sensors and the network in this situation.

B. Implementation

The PRISM model consists of three modules, one for the controller cluster, one for the network device layer, one for the sensor network. Part of the PRISM language description of the controller cluster is shown below.

```
// Part of the Controller Cluster module
    //1 = working, 0 = down
    c1 : [0..1] init 1;
    c2 : [0..1] init 1;
    // failure of control device
    [] c1>0 -> lambda_c1 : (c1' = 0);
    [] c2>0 -> lambda_c2 : (c2' = 0);
    [timeout1]  comp & (c1 > 0) & (c2 > 0)
                -> tau1 : true;
    [timeout1] !comp & (c1 > 0) & (c2 > 0)
                -> tau1 : true;
    [timeout2]  comp & (c1 > 0) & (c2 = 0)
                -> tau2 : true;
    [timeout2] !comp & (c1 > 0) & (c2 = 0)
                -> tau2 : true;
    [timeout3]  comp & (c2 > 0) & (c1 = 0)
                -> tau3 : true;
    [timeout3] !comp & (c2 > 0) & (c1 = 0)
                -> tau3 : true;
    [timeout4]  comp & (c2 = 0) & (c1 = 0)
                -> tau4 : true;
    [timeout4] !comp & (c2 = 0) & (c1 = 0)
                -> tau4 : true;
```

The formulae defining the relationship between the other two modules are shown below. When the number of devices

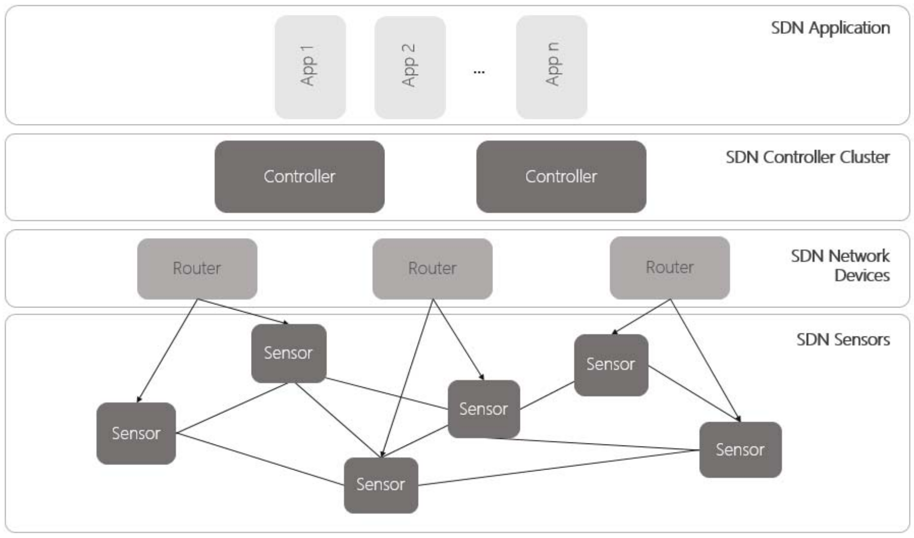

Figure 3. Architecture of the SDWSN.

connected to a certain router decreases, the failure rate of this router is decreased correspondingly.

```
//Relationship between the Network Device Layer
//and the Host Device Layer
//number of devices that a virtual switch
//can receive
formula num_h1 = NUM_H - net1;
formula num_h2 = NUM_H - net2;
formula num_h3 = NUM_H - net3;
//more host devices connected, more probability
//of transient fault or failure
formula net_t_1 = 1/(24*60*60 -
                floor (0.5 * net1));
formula net_f_1 = 1/(30*24*60*60 -
                floor (0.5 * net1));
formula net_t_2 = 1/(24*60*60 -
                floor (0.5 * net2));
formula net_f_2 = 1/(30*24*60*60 -
                floor (0.5 * net2));
formula net_t_3 = 1/(24*60*60 -
                floor (0.5 * net3));
formula net_f_3 = 1/(30*24*60*60 -
                floor (0.5 * net3));
```

V. RESULTS

This section analyzes the results of model checking using Prism on the formal model presented previously.

Figure 5 shows the failure probability of the two-controller SDWSN (in contrast, the failure probability of a one-controller SDWSN is shown in Figure 4). Failure rates of controllers vary in order to study how different pairs of failure rates influence the failure probability of the control plane. The following CSL statement is used for model checking:

```
P=? [ ! "down" U<=T "fail_cont" ]
```

From the model checking results, we have the following findings.

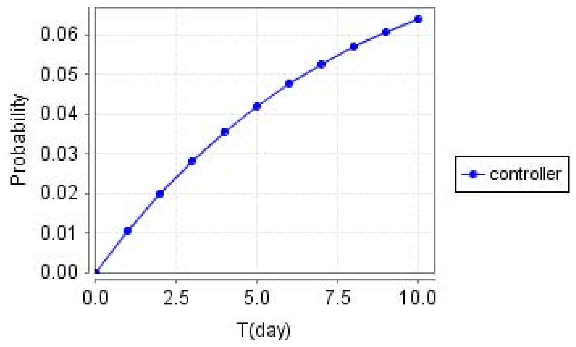

Figure 4. Failure probability of one controller. The failure rate is once in 90 days.

- The controller plane 45/45 is much better than the cluster 5/85.
- The controller plane 45/45, 30/60 and 20/70 are better than the one-controller control plane, even if the failure rate of the controller is once in 90 days.
- The controller plane 5/85 and 10/80 are worse than the one-controller control plane.

From these findings, we can conclude that two controllers are usually better than one controller. However, if one controller is unreliable, the reliability of the control plane will not increase significantly.

The failure probability of WSN layer in 10 hours is shown in Figure 6. The following CSL statement is used for model checking:

```
P=? [ !"down" U<=T "fail_h" ]
```

Here, ”fail_h” represents the failure of WSN layer. The failure probability in 10 day is shown in Figure 7. The two main

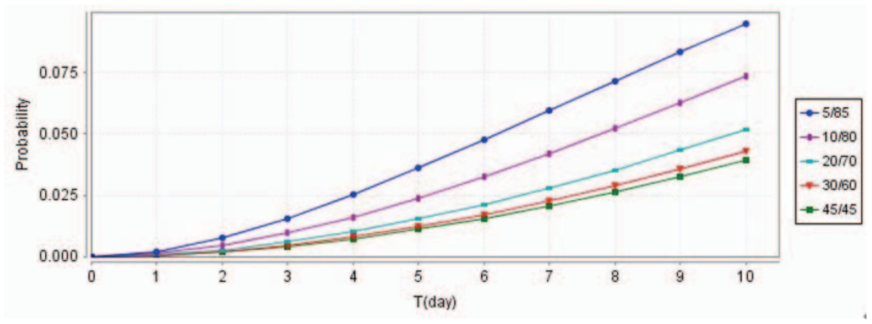

Figure 5. Failure probability of a pair of controllers. The failure rates of controllers are shown in the right.

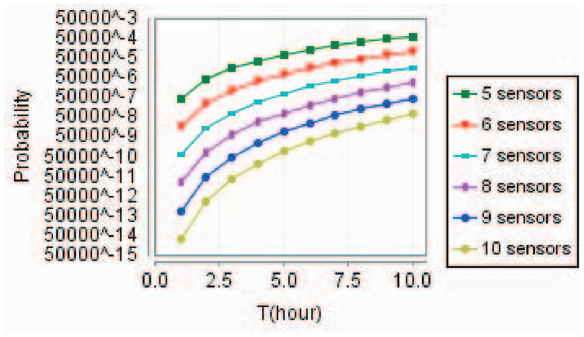

Figure 6. Probability of sensor layer in 10 hours. The number of sensors associated with each router is given in the right. For example, 5 sensors means that each router is associated with 5 sensors.

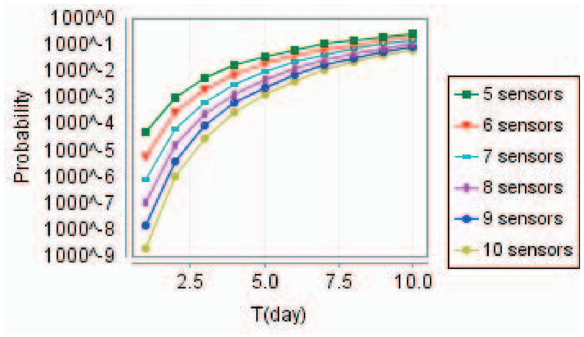

Figure 7. Probability of sensor layer failure in 10 days.

influential factors are time and the total number of sensors in the system.

- As time increases, the probability will increase.
- The probability drops as the total number of sensors increases. The reason is that with more sensors, it is less likely that all of them are down.

The result shows that increasing the number of sensors in the WSN is one way of increasing the reliability.

Figure 8 illustrates the probability of each failure type occurring first in an unlimited time. FC represents the failure due to the controller cluster; FH refers to the failure of WSN; and FN is the failure of routers. The result can be summarized as follows.

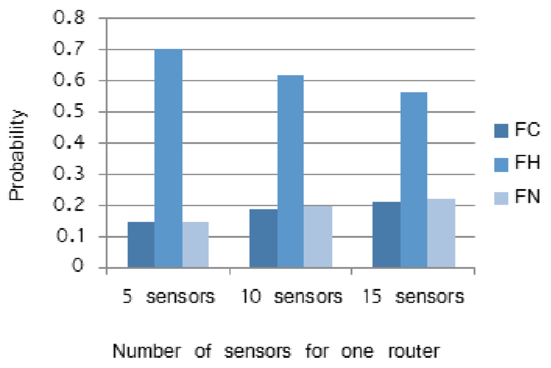

Figure 8. Probability of each failure type occurring first.

- It shows that sensor failure is the dominant failure type.
- With the number of sensors increase, FH decreases, however FN and FC increase.

Thus, if the number of sensors is too large, the system reliability may decrease, which is a trade-off.

VI. Related Work

Many researchers have studied the reliability of SDNs. Some researchers argued that the SDN controllers are not reliable enough under fatal disasters due to the decoupled control plane and data plane [3], [8]. As a result, SDN controllers are not ready for massive productions. To address such problem, [4], [7], [11] present several algorithms about the placement of controllers in order to maximize the reliability with multiple controllers. Apart from that, to detect bugs inside an SDN controller platform, a real-time post-deployment failure injection tool, named Chaos Monkey, is invented to increase the reliability of SDN [2]. Furthermore, ResilientFlow, An innovative approach is developed and applied to deal with unexpected link failures under large-scale SDNs [10]. The architecture of SDN controller is redesigned in [1] to be resilient to potential SDN application failures.

VII. Conclusion

In this paper, a formal SDWSN model is constructed and its properties are evaluated. CTMC is used to model the SDWSN;

and CSL is used for model checking SDWSN properties. In addition, some interesting findings are described below.

Factors that influence SDWSN reliability in our model are listed as follows: the number of central controllers, the failure rate of controllers, the number of sensors, the failure rate of routers and sensors.

One practical strategy of improving SDWSN reliability is using more that one controllers. Also, it would be better if all the controllers are relatively reliable. One unreliable controller may not increase the system reliability significantly.

Another practical strategy is increasing the number of sensor nodes. However, this may increase the failure rate of controllers and routers. Thus, designers are suggested to use a proper number of sensors which will not compromise controllers and routers.

Acknowledgment

This work has been supported in part by the XJTLU RDF140243, by the Natural Science Foundation of Jiangsu Province under Grant BK20150376, and by the Suzhou Science and Technology Development Plan under Grant SYG201516.

References

[1] Balakrishnan Chandrasekaran and Theophilus Benson. Tolerating sdn application failures with legosdn. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 22. ACM, 2014.

[2] Michael Alan Chang, Brendan Tschaen, Theophilus Benson, and Laurent Vanbever. Chaos monkey: Increasing sdn reliability through systematic network destruction.

[3] Xinjie Guan, Baek-Young Choi, and Sejun Song. Reliability and scalability issues in software defined network frameworks. In *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, pages 102–103. IEEE, 2013.

[4] Yannan Hu, Wendong Wang, Xiangyang Gong, Xirong Que, and Shiduan Cheng. On reliability-optimized controller placement for software-defined networks. *Communications, China*, 11(2):38–54, 2014.

[5] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *Formal methods for performance evaluation*, pages 220–270. Springer, 2007.

[6] Bruno Nunes, Manoel Mendonca, Xuan-Nam Nguyen, Katia Obraczka, Thierry Turletti, et al. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys & Tutorials, IEEE*, 16(3):1617–1634, 2014.

[7] Francisco Javier Ros and Pedro Miguel Ruiz. Five nines of southbound reliability in software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 31–36. ACM, 2014.

[8] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. Advanced study of sdn/openflow controllers. In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, page 1. ACM, 2013.

[9] William Stallings. Software-defined networks and openflow. *The Internet Protocol Journal*, 16(1):2–14, 2013.

[10] Takuma Watanabe, Takuya Omizo, Toyokazu Akiyama, and Katsuyoshi Iida. Resilientflow: Deployments of distributed control channel maintenance modules to recover sdn from unexpected failures. In *Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the*, pages 211–218. IEEE, 2015.

[11] Peng Xiao, Wenyu Qu, Heng Qi, Zhiyang Li, and Yujie Xu. The sdn controller placement problem for wan. In *Communications in China (ICCC), 2014 IEEE/CIC International Conference on*, pages 220–224. IEEE, 2014.