Software defined intelligent building

Rui Xue
Chinese Academy Of Sciences

Xin Huang
Xi'an Jiaotong University

Jie Zhang
University of Wollongong, jz248@uowmail.edu.au

Yulin Lu
Xi'an Jiaotong University

Ge Wu
University of Wollongong, gw523@uowmail.edu.au

See next page for additional authors

Software defined intelligent building

Abstract

The networks of intelligent building are usually consist of a great number of smart devices. Since many smart devices only support on-site configuration and upgrade, and communication between devices could be observed and even altered by attackers, efficiency and security are two key concerns in maintaining and managing the devices used in intelligent building networks. In this paper, the authors apply the technology of software defined networking to satisfy the requirement for efficiency in intelligent building networks. More specific, a protocol stack in smart devices that support OpenFlow is designed. In addition, the authors designed the lightweight security mechanism with two foundation protocols and a full protocol that uses the foundation protocols as example. Performance and session key establishment for the security mechanism are also discussed.

Keywords

building, software, intelligent, defined

Disciplines

Engineering | Science and Technology Studies

Publication Details

Xu, R. Yang., Huang, X., Zhang, J., Lu, Y., Wu, G. & Yan, Z. (2015). Software defined intelligent building. International Journal of Information Security and Privacy, 9 (3), 84-99.

Authors

Rui Xue, Xin Huang, Jie Zhang, Yulin Lu, Ge Wu, and Zheng Yan

Software Defined Intelligent Building

Rui Yang Xu, Xi'an Jiaotong-Liverpool University, Suzhou, China

Xin Huang, Xi'an Jiaotong-Liverpool University, Suzhou, China

Jie Zhang, Xi'an Jiaotong-Liverpool University, Suzhou, China

Yulin Lu, Xi'an Jiaotong-Liverpool University, Suzhou, China

Ge Wu, University of Wollongong, Australia

Zheng Yan, Aalto University, Espoo, Finland & Xidian University, China

ABSTRACT

The networks of intelligent building are usually consist of a great number of smart devices. Since many smart devices only support on-site configuration and upgrade, and communication between devices could be observed and even altered by attackers, efficiency and security are two key concerns in maintaining and managing the devices used in intelligent building networks. In this paper, the authors apply the technology of software defined networking to satisfy the requirement for efficiency in intelligent building networks. More specific, a protocol stack in smart devices that support OpenFlow is designed. In addition, the authors designed the lightweight security mechanism with two foundation protocols and a full protocol that uses the foundation protocols as example. Performance and session key establishment for the security mechanism are also discussed.

Keywords: *Intelligent Building, OpenFlow, Security, Smart Devices, Software Defined Networks*

1. INTRODUCTION

An intelligent building (Bo et al., 2014; Huang et al., 2015; Zhang et al., 2014; Li et al., 2013; Son et al. 2011; Bruno et al., 2014) can monitor and control security, lighting, fire safety and many other systems in a building. It can significantly improve occupant comfort, simplify operation of building systems, and reduce energy consumption and operating costs.

The foundation of intelligent building is the large number of smart devices. These devices are usually connected, and form a very large network. The network is named as *intelligent building network* (IBNet) in this paper.

DOI: 10.4018/IJISP.2015070105

However, one challenge here is the IBNet management. So far, many smart devices only support on-site configuration and upgrade. Consequently, it is time-consuming to update configurations (e.g., routing policies and security rules) one device by one device.

In this paper, the researchers propose software defined networking (SDN)-based IBNet to eliminate the problem mentioned above. Recent years, SDN (Bruno et al. 2014; Kretutz et al., 2013; Lantz et al. 2010; Drutskoy et al., 2013; Handigol et al., 2012) is proposed as the next generation network technology, and it can be easily managed. Thus, the researchers expect that with this new technology, the IBNet can also be easily managed. However, the research questions are:

1. Is SDN architecture proposed for general networks suitable for IBNet?
2. Can protocol stacks in smart devices be extended to support SDN protocols?
3. How can the researchers balance requirements of security (Huang et al. 2015) and performance in the newly designed network?

In order to study these questions, the authors build a prototype: SBuilding. The prototype is inspired from SDN: network devices become programmable in SBuilding. SBuilding is compared to other networking technologies, the advantages are summarized.

The main problem in SBuilding is OpenFlow (McKeown et al. 2008; Vu et al., 2012). OpenFlow is the most widely used protocol suite between a SDN controller and network devices. However, the standard version of OpenFlow does not support smart devices.

In order to solve this problem, the authors have designed a protocol stack in smart devices to support OpenFlow. Also, the authors have integrated lightweight security mechanisms in the extended OpenFlow to secure the communications. Both theoretical analysis and practical evaluation are used to confirm that the performance of these mechanisms are acceptable.

In summary, main contributions of this paper are as follows:

1. A SDN-based IBNet (SNet) for intelligent building is proposed. Its advantages and disadvantages comparing to several current network architectures are analyzed;
2. OpenFlow is extended to support smart devices in intelligent buildings. Its security mechanisms are also designed;
3. The researchers use the model checking approach, which is a formal verification technology, to verify proposed security mechanisms;
4. Performance of extended OpenFlow is studied; the advantages and disadvantages are summarized.

This paper is organized as follows. In Section II, SNet is introduced. In Section III, lightweight security mechanisms are explained. Section IV and Section V discuss the security and performance of proposed protocols respectively. Section VI shows a use case of our system. Section VII gives a key management scheme for distributed the keys. Finally, a conclusion is made in Section VIII.

2. SYSTEM OVERVIEW

Here, SDN technologies are used in IBNets. The newly designed network, which is named as SDN-based IBNet (SNet), aims at providing an easily manageable network for intelligent buildings.

2.1. SNet

SNet has a layer-based architecture, which is shown in Figure 1. The responsibilities of each layer are introduced below.

- **Application plane:** This top layer is comprised of applications that are responsible for network management. These applications can monitor and manage the whole network with the help of the control plane (explained below) (Vu et al., 2012);
- **Control plane:** This plane can collect information from the data plane (explained below). It can also manage smart devices in the data plane according to application requirements;
- **Data Plane:** The data plane is a network of smart devices. However, it only has the capability of packet transmission. All the behaviors of smart devices should conform to the rules defined by the controller (or rather applications);
- **Southbound API:** The controller communicates with the data plane through the southbound API. One of the implementation of the Southbound API is OpenFlow;
- **Northbound API:** Applications communicate with the control plane using the northbound API. Examples include REST API provided by OpenDayLight and FloodLight.

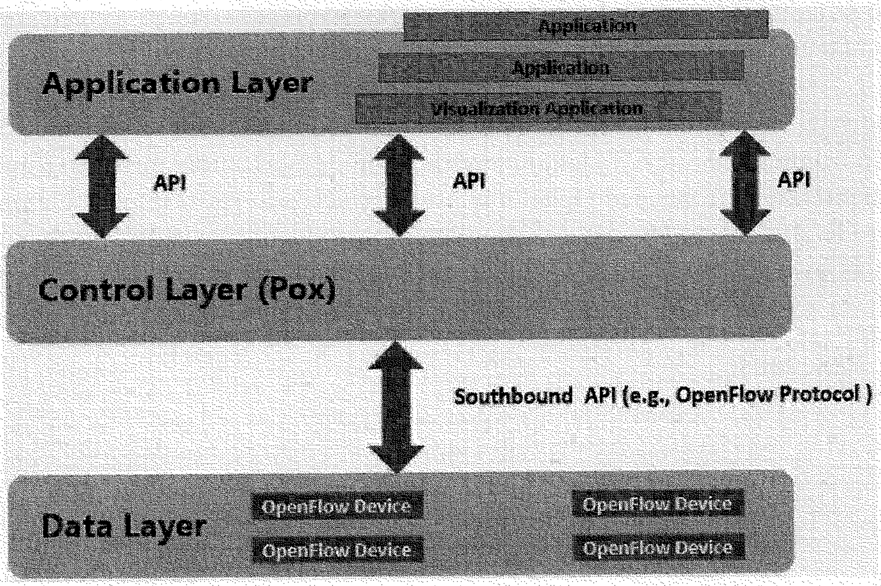
2.2. OpenFlow Device

The extended OpenFlow allows the controller to communicate with smart devices. The smart device implemented the extended OpenFlow protocol stack is named as OpenFlow device. The stack consists of two components: OpenFlow agent and flow table, which are explained below.

2.2.1. OpenFlow Agent

An OpenFlow agent in an OpenFlow device is responsible to communicating with the controller, handling packets received from the controller or other nodes. Below is the algorithm used in the OpenFlow agent.

Figure 1. Snet architecture



```
OpenFlow agent process procedure
initialize OpenFlow connection
while(connection established)
{
      parse received packet
      if(packet is an OpenFlow message)
      {
            fulfill it into flow table
       }
        else
      {
         Find a match for the corresponding packet by looking
up flow entries defined in flow table
            if(a match exists)
           {
             apply corresponding action to that packet
            }
            else
           {
            send it to the controller
            }
       }
}
```

2.2.2. Flow Table

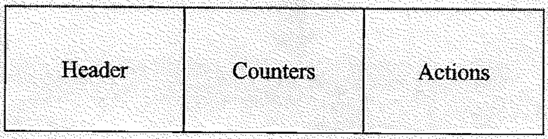
The flow table contains a sequence of flow entries that is received from the controller. A flow entry is depicted in Figure 2. It has three main fields: Header, Counters and Actions. (1) Header is a set of quantifies that are used for identify packet types. (2) Counters are used to record the lifetime of the packet entry. (3) Actions specify actions that will be applied to the packets.

2.3. Light Weight Security Mechanisms

SNets are different from general computer networks. In order to design security mechanisms for extended OpenFlow, the authors need to review requirements firstly:

1. **Integrity:** Messages transmitted between controller and OpenFlow devices should be protected from unauthorized deletion, modification or fabrication during transmission;
2. **Authentication:** Extended OpenFlow protocol needs to resist impersonation attack. Both controller and OpenFlow devices should be able to identify whether the received messages are sent by the real source or an impersonation attacker;

Figure 2. Flow entry



3. **Lightweight:** (1) Many smart devices do not have enough computing resource to run asymmetric cryptographic algorithms. (2) Energy is a big concern in smart devices, thus expensive computations should be avoided to prolong its lifetime.

In summary, lightweight security mechanisms are required to guarantee message integrity and authenticity in extended OpenFlow protocol, which will be introduced below.

3. LIGHTWEIGHT SECURITY MECHANISMS

In this section, lightweight security mechanisms in extended OpenFlow protocol are introduced. Two foundation protocols are described firstly, and one example protocol is provided to show the usage of the foundation protocols.

3.1. Foundation Protocols

An OpenFlow protocol normally contains several steps of communication between controller and OpenFlow device in one session. In order to achieve the aforementioned requirements, the researchers use message authentication code (MAC) at the last step of communication.

The symbols used afterwards are explained in Table 1. The protocol assumption is: k is a pre-shared key held by the controller and the OpenFlow device in one session.

3.1.1. Protocol I: OpenFlow Device Authenticates Mc and Mc'

Step 1: Device chooses a nonce Nd and sends <Nd, D> to controller.
Step 2: Controller computes MAC=HMAC (k, Nd||C||D|| Mc||Mc'|| Md), and then sends <C, Mc', MAC> to the OpenFlow device.
Step 3: Device verifies the received MAC through computing HMAC (k, Nd||C||D||Mc||Mc'||Md) itself. If the verification succeeds, the device can confirm that: (1) Mc is really sent from controller in this session; (2) the device and controller have agreed on Mc and Mc'.

Table 1. Symbols in foundation protocols

Symbol	Meaning
k	k is a shared key held by the controller and the OpenFlow device in one session
C	identity of controller
D	identity of OpenFlow device
Mc	the whole messages sent by controller during the previous steps of communication in this session
Md	the whole messages sent by OpenFlow device during the previous steps of communication in this session
Mc'	the last piece of message sent by controller to device
Md'	the last piece of message sent by device to controller

3.1.2. Protocol II: Controller Authenticates Md and Md'

Step 1: Controller chooses a nonce Nc and sends <Nc, C> to the OpenFlow device.
Step 2: Device computes MAC=HMAC (k, Nc||D||C||Md||Md'||Mc), and sends <Md', D, MAC> to controller.
Step 3: Controller verifies the received message by computing HMAC (k, Nc||D||C||Md||Md'||Mc) and checking if the result equals the received MAC.

3.2. Example Protocol

In order to show how extended OpenFlow is used, one example protocol is provided. This protocol is the initialization procedure of extended OpenFlow, some symbols are explained in Table 2.

3.2.1. Protocol III: Extended OpenFlow Initialization

Step 1: Controller chooses a nonce Nc and sends <Hello, Nc, C> to the OpenFlow device.
Step 2: Device chooses a nonce Nd and sends <Hello, Nd, D> to controller.
Step 3: Controller sends Feature Request to the OpenFlow device.
Step 4: OpenFlow device responses controller with Feature Response.
Step 5: Controller sends device SetConfig.
Step 6: Controller computes MAC1=HMAC(k, Nc$\oplus$ Nd$\oplus$ (C||D||Mc||Mc'||Md)) and sends OpenFlow device with MAC1, Barrier Request and Flow Mod.
Step 7: OpenFlow device verifies the received MAC1. If the verification succeed, i.e. < Nc, Nd, Mc, Md, Mc'> have not been modified during transmission, the device computes MAC2 through the following formula and sends MAC2 with Barrier Response to controller:

MAC2 = HMAC(k, Nd $\oplus$Nc $\oplus$ (D||C||Md||Md'||Mc||Mc'))

Controller can verify the received MAC2 to make sure the integrity and authentication of the entire received message, that is < Nc, Nd, Mc, Md, Mc', Md'>.

4. SECURITY ANALYSIS

The authors discuss the security of this protocol from three aspects: integrity, authentication and replay attack as follows.

Table 2. Symbols in the example protocol

Symbol	Meaning
Mc	Hello||Feature Request||SetConfig
Md	Hello||Feature Response
Mc'	Barrier Request||Flow Mod
Md'	Barrier Response

4.1. Integrity

Integrity here means the whole messages transmitted between controller and OpenFlow device have not been modified by attackers. This is guaranteed by MAC in the last steps. Since the key is held only by controller and OpenFlow device, only these two parties can compute this MAC. If any message were modified by attackers, then the verification for MAC (MAC1, MAC2) will fail.

4.2. Authentication

These protocols provide authentication through the application of nonce and message authentication code so that an attacker cannot successfully impersonate either the controller or the device. Since the key is held only by controller and OpenFlow device, only these two parties can compute the MAC corresponding to the specific nonce in each session.

4.3. Replay Attack

Thanks to the long nonce, replay attacks can be eliminated. Take Protocol I as the example. As long as Nd is long enough, it is less likely that current Nd was used recently. Consequently, the attacker finds it difficult to reuse the recorded MACs in previous sessions (using different Nd). For example, the probability of using the same 1024-bit nonce in two consecutive sessions is 1/2^1024.

4.4. Security Evaluation

In order to validate our Extended OpenFlow protocol, the authors use model checking method to verify their authenticity of this protocol. Firstly, the authors re-write it as follows:

1. C →D: C, Nc, Mc
2. D →C: D, Nd, Md
3. C →D: C, hash (k, Nc, Nd, C, Mc, D, Md)
4. D →C: D, hash (k, Nd, Nc, D, Md, C, Mc)

The verification result is shown in Figure 3. No attack was found.

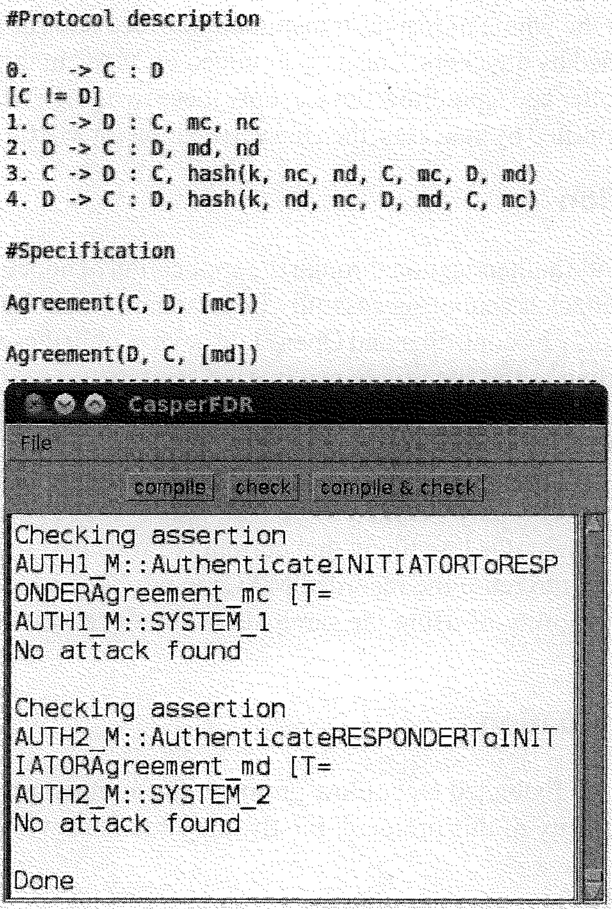
One note is that the sequence of the identities in the hash (or MAC generation function) is important. If message 3 and 4 are write as below:

1. C →D: C, hash (k, Nc, Nd, C, Mc, D, Md)
2. D →C: D, hash (k, Nd, Nd, C, Mc, D, Md)

The attack below is possible:

1. C →I_D: C, Nc, Mc
2. I_D →C: D, Nc, I_Md
3. C →I_D: C, hash (k, Nc, Nc, C, Mc, D, I_Md)
4. I_D →C: D, hash (k, Nc, Nc, C, Mc, D, I_Md)

Figure 3. The verification result



```
#Protocol description

0.    -> C : D
[C != D]
1. C -> D : C, mc, nc
2. D -> C : D, md, nd
3. C -> D : C, hash(k, nc, nd, C, mc, D, md)
4. D -> C : D, hash(k, nd, nc, D, md, C, mc)

#Specification

Agreement(C, D, [mc])

Agreement(D, C, [md])
```

```
Checking assertion
AUTH1_M::AuthenticateINITIATORToRESP
ONDERAgreement_mc [T=
AUTH1_M::SYSTEM_1
No attack found

Checking assertion
AUTH2_M::AuthenticateRESPONDERToINIT
IATORAgreement_md [T=
AUTH2_M::SYSTEM_2
No attack found

Done
```

Here, I_D is an attacker, and I_Md is its fake commands. Since the hash inputs in message 3 and 4 are the same, the attacker can just copy the hash output in message 3, and reply it in message 4.

However, if the researchers change the sequence of identities in the two inputs, the attacker must compute the hash output by himself. This is difficult because the attacker does not have the shared key.

5. PERFORMANCE ANALYSIS

In order to analyze the performance of these new proposed protocols, the authors consider three factors: the length of messages in every step and computation cost in OpenFlow device and controller respectively.

Denote M the pieces of transmitted message, H the HMAC algorithm, LMc' the length of Mc' and LMd' the length of Md' in byte. Table 3 shows the performance of Protocol I and II, and in Table 4 the researchers compare the performance of our Extended OpenFlow protocol with the original unsecured OpenFlow protocol.

From Table 4 we can see that the total length of message is only 64 byte longer than the unsecured protocol, and the computation cost in device and controller is computing two HMAC.

Table 3. Performance of Protocol I and II

Parameters	Protocol I	Protocol II
message in step 1	16 bytes	16 bytes
message in step 2	LMc'+40 bytes	LMd'+40 bytes
Computation cost in Device	H	H
Computation cost in Controller	H	H

Table 4. Performance of Protocol III

Parameters	Secured Protocol	Unsecured Protocol
Message in step 1	24 bytes	8 bytes
Message in step 2	24 bytes	8 bytes
Message in step 3	8 bytes	8 bytes
Message in step 4	8 bytes	8 bytes
Message in step 5	12 bytes	12 bytes
Message in step 6	144 bytes	112 bytes
Message in step 7	40 bytes	8 bytes
Computation cost in Device	2H	0
Computation cost in Controller	2H	0

Since the generation algorithm HMAC neither requires much computation resource nor consume much energy, the additional cost of our secured OpenFlow protocol is accessible.

6. IMPLEMENTATION AND EVALUATION

In this section, our demo system SBuilding is introduced. Also, the OpenFlow initialization procedure is evaluated in this demo system.

6.1. SBuilding

SBuilding is a demo intelligent building system, which implemented SNet. It consists of four parts: smart devices, central controller, mobile controller, and a cloud server, which is shown in Figure 4.

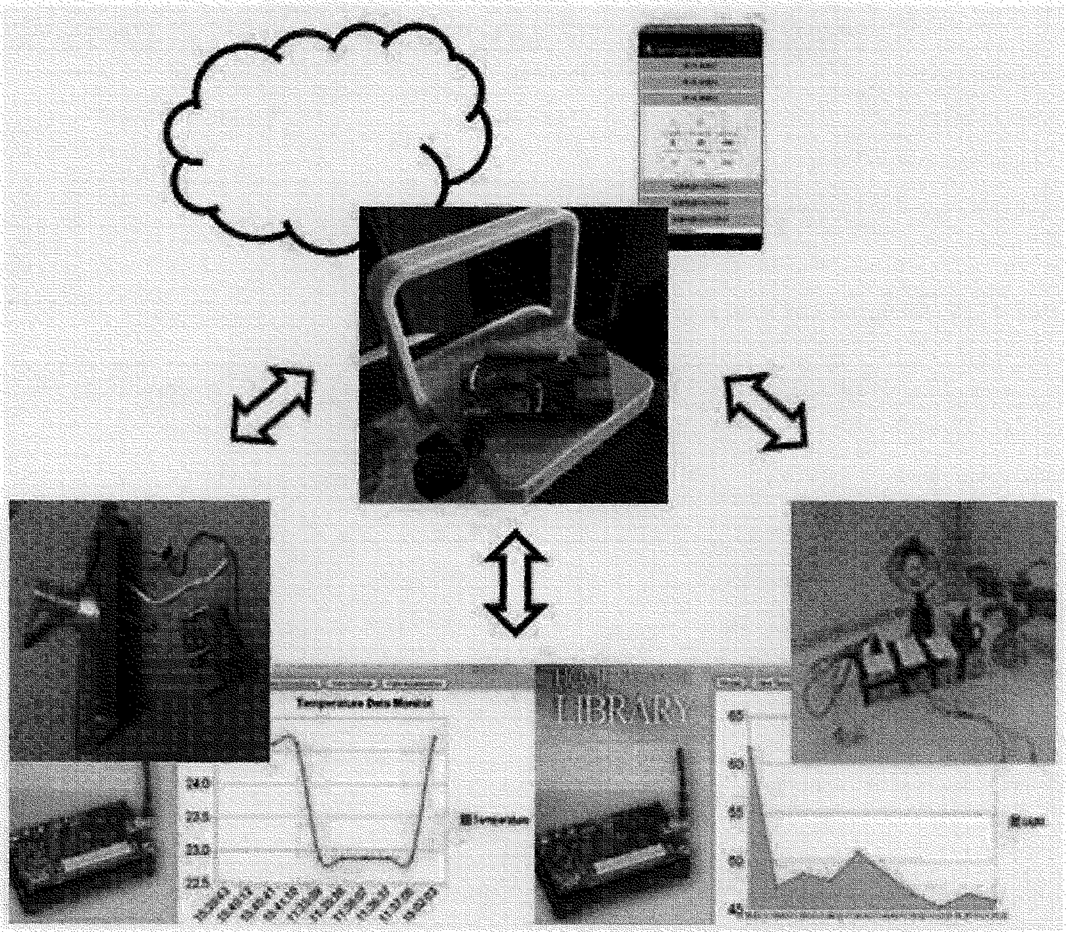
Smart devices include smart sockets, smart lock, and many others. These devices are able to communicate with the central controller.

Central controller is used to control these smart devices. It can also work as a bridge between mobile controller/cloud server and smart devices.

Mobile controller is a mobile phone application. It can communicate with smart devices via the central controller.

Cloud server is mainly used to store historical data. It can also relay messages between the central controller and the mobile controller in case users want to control smart devices remotely.

Figure 4. SBuilding



6.2. OpenFlow in SBuilding

Once an OpenFlow device receives a valid message from controller, it will interpret the message by examining the 'type' field and actions will be taken correspondingly. For instance, the device shall calculate and send MAC2 after receiving MAC1 from the controller. A successful running procedure of Protocol III is shown in Figure 5.

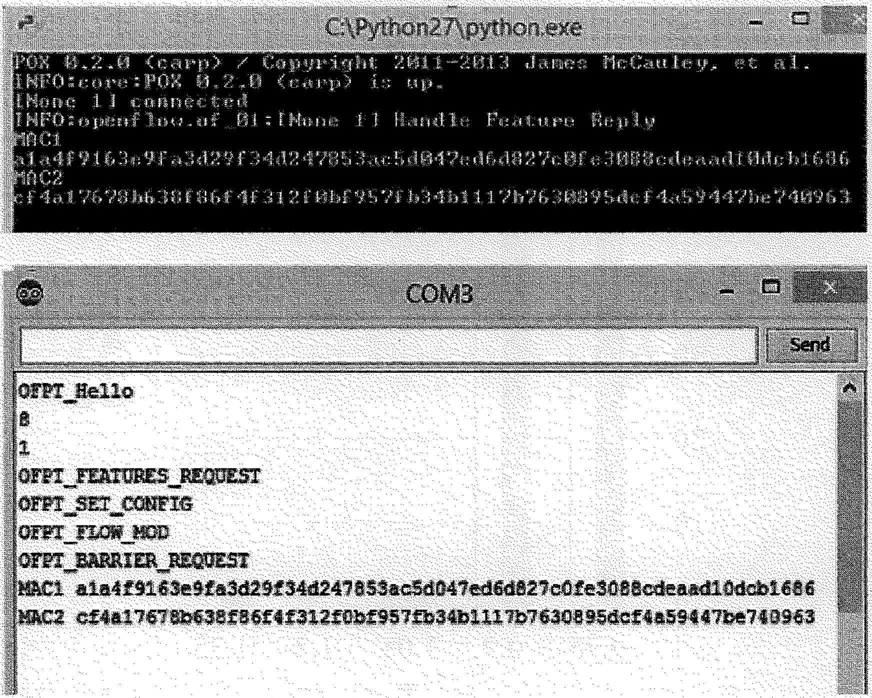
6.3. Performance Evaluation

The performance of the initialization procedure (Protocol III) is evaluated in SBuilding. The controller is connected to Arduino. The controller is Pox, a centralized control platform for research and education. The OpenFlow device platform is Arduino Uno, a fundamental and robust toolset.

To measure how much time elapses during the process of secured OpenFlow protocol on Arduino, a counter could be activated at the end of each time period. The overall description of this experiment is summed as Table 5 and Figure 6. Two sets of experiments are conducted, including Secured OpenFlow and Unsecured OpenFlow initializations. For both of them, the memory consumption and each running time are recorded for later analysis.

From Table 5 and Figure 6 we can see that our secured OpenFlow provides two-part authentication with only increments of 0.024s in average time of TTL and 32bytes in memory. The

Figure 5. Implementation of Protocol III



```
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
[None 1] connected
INFO:openflow.of_01:[None 1] Handle Feature Reply
MAC1
a1a4f9163e9fa3d29f34d247853ac5d047ed6d827c0fe3088cdeaad10dcb1686
MAC2
cf4a17678b638f86f4f312f0bf957fb34b1117b7630895dcf4a59447be740963
```

```
OFPT_Hello
8
1
OFPT_FEATURES_REQUEST
OFPT_SET_CONFIG
OFPT_FLOW_MOD
OFPT_BARRIER_REQUEST
MAC1 a1a4f9163e9fa3d29f34d247853ac5d047ed6d827c0fe3088cdeaad10dcb1686
MAC2 cf4a17678b638f86f4f312f0bf957fb34b1117b7630895dcf4a59447be740963
```

Table 5. Experiment parameters

Parameters	Secured OpenFlow Initialization	Unsecured OpenFlow Initialization
Smart device	Arduino Uno	Arduino Uno
Controller	Pox	Pox
Number of rounds	100	100
Average time of TTL	0.8385s	0.8145s
Memory footprint	16.254kbytes	15.667kbytes

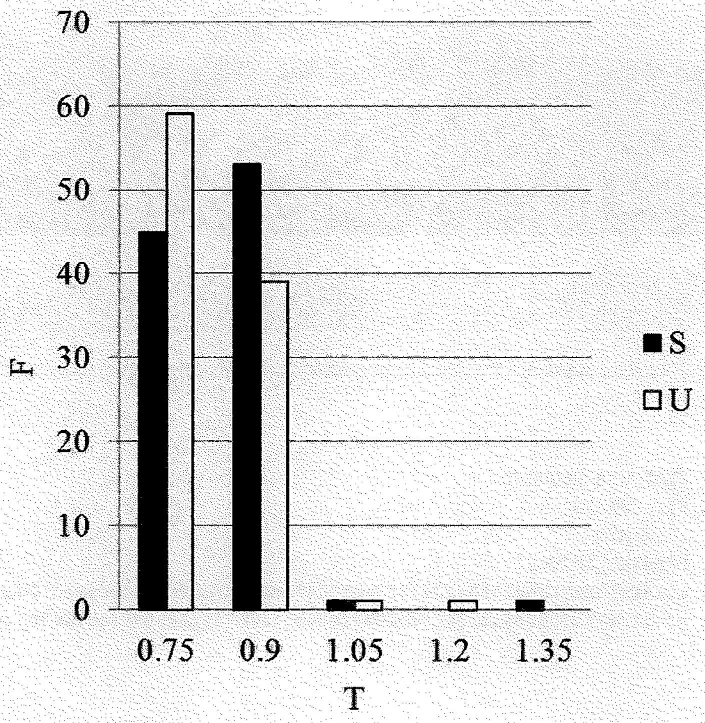
histogram for the running time of each initialization (RTI) is depicted in Figure 6, in which S and U denote Secured and Unsecured OpenFlow Initializations respectively. T and F represent running time and frequencies for both initializations. The results indicate that 98 percent of the RTI falls in the range of [0.75, 0.9] with 2 percent exceptions. Overall, the running time of S is as stable as U.

7. COMPARISONS

7.1. Advantages

The network can gain various benefits by separating the control and data planes. The advantages are described as follows:

Figure 6. Distribution of running time



- **Flexibility:** With the introduction of the programmability of the network, routing rules can be easily modified. In this case, the routing policy within IBNet could dynamically modified in different scenarios;
- **Global View:** Since a controller has a bird eye view of the whole network, it can make better decision for the whole network. For example, the controller can easily select the least congested path since it knows all the link status;
- **Security:** The status of the whole network is monitored via controller applications, which allow better reaction to attacks. Also, the network manager can reprogram the smart devices to stop attacks, or at least reduce the impact of attacks. For example, when one device has been compromised, the controller can reprogram nodes nearby so that these nodes will not send/receive messages to/from the compromised node.

7.2. Comparing to ZigBee-like Architecture

One IBNet architecture of intelligent building is ZigBee-like architecture (Duan, & Li, 2008; Paolo et al. 2007).

ZigBee nodes do not have the globe network information. Thus the routing algorithm decision is not always optimized for the whole network. Now, the controller has a global view, thus better routing decisions can be made. For example, the controller can switch from the most energy efficient route policy to the longest live route policy (i.e. achieve a balance between routing nodes with more energy left and nodes with less energy left) when node death can lead to serious outcome.

Also, if a node is compromised, the controller can isolate this node immediately. This is achieved by reprogramming the nearby nodes, and stop communications between the compromised nodes and these neighbor nodes.

7.3. Comparing to Bluetooth-like Architecture

Bluetooth network architecture is one option for IBNets. Bluetooth master has a globe view of the whole network. Also, Bluetooth supports lightweight security mechanisms. However, Bluetooth-like architecture only supports star topology, which is quite limited in a complex IBNet. Also, it is not easy to reconfigure routing or security policies.

7.4. Comparing to WiFi-like Architecture

Another IBNet architecture of intelligent building is WiFi. According to Paolo et al. (2007), all the wireless sensor nodes and servers are connected to access points (AP). AP does not have the globe view, thus it is difficult to make good routing decisions regarding the whole network. Also, WiFi does not support smart device reconfiguration. Additionally, WiFi security mechanisms are usually heavier than ZigBee and Bluetooth.

8. DISCUSSION OF SESSION KEY ESTABLISHMENT

In the establishment of our Protocol I and Protocol II, a symmetric key k is assumed to be pre-shared by controller and device in one session. Here the researchers give a key agreement approach for the generation of k.

Assume mk is the master key that can be used to generate k for controller and the device in every session. The master key can be pre-installed in the factory or sent out-of-band from a trust centre. Denote symmetrically encrypting message M under key mk by $\{M\}_{mk}$, key derivation function by f(a, b) where f(·) could be bitwise XOR of the inputs or application of a hash function to the concatenation of the inputs. The follow scheme shows how to establish the symmetric key k from mk.

8.1. Protocol IV: Key Agreement

1. Controller generates a nonce Nc and sends it to device;
2. Device generates a nonce Nd and a random value Rd, encrypts Nd, Nc, C, Rd under mk and sends the cyphertext $\{Nd, Nc, C, Rd\}_{mk}$ to controller;
3. Controller decrypts $\{Nd, Nc, C, Rd\}_{mk}$ and checks Nc. If the verification succeed, controller choose a random value Rc and send $\{Nd, Nc, Rc\}_{mk}$ to device;
4. Device decrypts $\{Nd, Nc, Rc\}_{mk}$ and verifies Nd;
5. Controller and device compute the shared symmetric key k=f(Rc, Rd) respectively.

This key agreement protocol is from a sever-less key establishment mechanism of international standard ISO/IEC 11770-2, which has been proved to be secure.

8.2. Security Analysis

Security properties with brief analysis of Protocol IV are listed as follows.

The adversary is not able to initiate impersonation attacks. This is because the adversary does not have the master key mk, which is used for authentication through encrypting and decrypting all the messages exchanged between controller and OpenFlow device.

The adversary is not able to derive the generated symmetric key k. The symmetric key is computed from the two random values Rd and Rc. Since Rc and Rd are transmitted in the form of ciphertext that only parties holding mk can decrypt them and acquire the plaintext, the attacker cannot get Rd and Rc as long as the encryption algorithm applied in this protocol is secure. Therefore, the adversary cannot compute k.

Forward secrecy. Suppose a previous symmetric key k1 is disclosed, the adversary is not able to derive the current symmetric key k2 from k1. The reason is that computation of the symmetric key in different runs of the protocol is independent. Knowing k1 gives no advantage to the adversary to computing k2=f(Rc', Rd').

8.3. Performance Analysis

Denote M the messages transmitted, E the AES encryption algorithm and D the AES decryption algorithm. The following table (Table 6) is a performance analysis considering both communication and computational costs of Protocol IV.

9. CONCLUSION

An intelligent building uses digital technologies to enhance performance and wellbeing, to reduce costs and resource consumption, and to engage more effectively and actively with its residents. IBnets should satisfy both efficiency and security for intelligent building systems. In this paper, a SDN based IBNet is designed and the OpenFlow in SDN is extended to support smart devices in IBnet.

In order to satisfy the requirement of security, the researchers have designed security mechanism and proposed several security protocols. Security and performance of these protocols have been discussed.

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.

Table 6. Communication cost and computational cost in each run of the protocol

	Communication Cost	Computational Cost
Controller	2 M	1 E+2 D
Device	2 M	1 E+2 D
Total cost	4 M	2 E+24 D

REFERENCES

Baronti, P., Pillai, P., Chook, V. W. C., Chessa, S., Gotta, A., & Hu, Y. F. (2007). Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and ZigBee standards. *Computer Communications*, *30*(7), 1655–1695. doi:10.1016/j.comcom.2006.12.020

Bo, W., Zhang, Y., Hong, X., Sun, H., & Huang, X. (2014). Usable security mechanisms in smart building. *Proceedings of the 2014 IEEE 17th International Conference on Computational Science and Engineering (CSE)* (pp. 748–753). IEEE.

Drutskoy, D., Keller, E., & Rexford, J. (2013). Scalable network virtualization in software-defined networks. *IEEE Internet Computing*, *17*(2), 20–27. doi:10.1109/MIC.2012.144

Duan, P., & Li, H. (2008). Zigbee wireless sensor network based multi-agent architecture in intelligent inhabited environments. *Proceedings of the 2008 IET 4th International Conference on Intelligent Environments* (pp. 1-6). IET.

Handigol, N., Heller, B., Jeyakumar, V., Maziéres, D., & McKeown, N. (2012). Where is the debugger for my software-defined network? *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 55–60). ACM. doi:10.1145/2342441.2342453

Huang, X., Bo, W., Zhang, Y., & Gong, N. (2015). I-lock: A phone-based access control system. *Proceedings of the International Conference on Computing and Technology Innovation (CTI 2015).*

Huang, X., Craig, P., Lin, H., & Yan, Z. (2015). *Seciot: a security framework for the internet of things.* Security and Communication Networks.

Kreutz, D., Ramos, F. M. V., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14–76. doi:10.1109/JPROC.2014.2371999

Lantz, B., Heller, B., & McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM. doi:10.1145/1868447.1868466

Li, J., & Zhang, Y. (2013). Intelligent building automation and control based on indasibms. *Proceedings of the 2013 International Conference on Service Sciences (ICSS)* (pp. 266–270). IEEE.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., & Turner, J. et al. (2008). Openflow: Enabling innovation in campus networks. *Computer Communication Review*, *38*(2), 69–74. doi:10.1145/1355734.1355746

Nunes, B. A. A., Mendonca, M., Nguyen, X.-N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials*, *16*(3), 1617–1634. doi:10.1109/SURV.2014.012214.00180

Son, J. Y., Park, J. H., Moon, K. D., & Lee, Y. H. (2011). Resource-aware smart home management system by constructing resource relation graph. *IEEE Transactions on* Consumer Electronics, *57*(3), 1112–1119.

Vu, T. H., Nam, P. N., Thanh, T., Linh, N. D., Thien, T. D., & Thanh, N. H. (2012). Power aware openflow switch extension for energy saving in data centers. *Proceedings of the 2012 International Conference on Advanced Technologies for Communications (ATC)* (pp. 309–313). IEEE.

Zhang, L., Liu, B., Tang, Q., & Wu, L. (2014). The development and technological research of intelligent electrical building. *Proceedings of the 2014 China International Conference on Electricity Distribution (CICED)* (pp. 88–92). IEEE.

Zheng Yan (M'06, SM'14) received the BEng degree in electrical engineering and the MEng degree in computer science and engineering from the Xi'an Jiao tong University, Xi'an, China in 1994 and 1997, respectively, the second MEng degree in information security from the National University of Singapore, Singapore in 2000, and the licentiate of science and the doctor of science in technology in electrical engineering from Helsinki University of Technology, Helsinki, Finland in 2005 and 2007. She is currently a professor at the Xidian University, Xi'an, China and a visiting professor at the Aalto University, Espoo, Finland. She authored more than 100 publications and solely authored two books. She is the inventor and co-inventor of 43 patents and patent applications. Her research interests are in trust, security and privacy, social networking, cloud computing, networking systems, and data mining. Prof. Yan serves as an editor/guest editor, an organization and program committee member for numerous international journals, conferences and workshops. She is a senior member of the IEEE.