Flexible, direct interactions between CoAP-enabled IoT devices

Girum Ketema Teklemariam, Jeroen Hoebeke, Ingrid Moerman, Piet Demeester
Department of Information Technology
Gent University - iMinds
Gent, Belgium
{firstname.lastname}@intec.ugent.be

Abstract— *The wireless communication capability of sensors and actuators made them suitable for several automation solutions which involve sensing physical properties and acting upon them. These days, gateway or cloud based sensor/actuator interaction models are widely used. In this model, every sensor/actuator interaction goes through the gateway or via the cloud. In order to realize the true Internet of Things philosophy where everything is interconnected, direct interactions between sensors and actuators, also called bindings, are important. In this paper, we introduce a CoAP based sensor/actuator binding solution where a 3rd party is responsible for setting up the binding, but is not involved in any of the further interactions between the sensor and actuator. As binding creation and execution is fully based on RESTful CoAP interactions, very flexible bindings between any two devices can be created. We implemented this solution in Contiki and evaluated the implementation by taking different measures such as delay, memory footprint, and packet size.*

Keywords – IoT; CoAP; Sensor/Actuator Binding

I. INTRODUCTION

Sensors and actuators have been in use in several gadgets that we see in our daily life including motion activated surveillance cameras, automatic air conditioning systems, and automatic doors. With the recent advances in electromechanical technologies, these sensors and actuators have been equipped with wireless communication capability and have become part of networked systems, such as wireless home and industry automation systems [1], automated industrial process monitoring systems [2] and smart grid [3]. In these systems, sensors usually collect data from the physical world and deliver it to a central service that, if needed, acts upon actuators. In a wireless home automation system, for example, a switch, when pressed, sends wireless signals to the bulb so that it takes the required action (turn on or turn off the light). In this example, the sensor is associated with the switch and the actuator is associated with the light bulb. Such systems were usually managed by proprietary controllers. This way, every interaction between sensor and actuator nodes was only possible through these managers using a pre-defined user interface. This approach is extremely closed with no room for customization and has severe compatibility issues if integration with other systems is a necessity.

The introduction of open standards such as 6LoWPAN [4] and CoAP [5], [6] opened a whole new possibility for the sensor/actuator networks by enabling direct access to the nodes and the resources associated with them from the Internet. 6LoWPAN introduced an adaptation layer so that IPv6 packets can be successfully transmitted through a wireless sensor and actuator network. CoAP, the lightweight counterpart of HTTP, allows using web services to interact with sensor and actuator nodes. The CoAP draft specifies how the protocol can be used to achieve embedded web services on constrained devices such as sensor and actuator nodes, and defines a mapping with the well-known HTTP protocol. These new protocols allow users from the Internet to interact directly with the constrained nodes using different applications, just as they interact with any other web service running over the Internet.

In this paper, we will show how the CoAP protocol and the observe option (along with the conditional observe extension)[10], [11] can be used to create direct associations, also called *bindings*, between sensors and actuators in a flexible way. The main contribution of this paper is a novel solution that enables direct interactions between sensors and actuators, eliminating the need for external devices to continuously coordinate communication between them. The interactions are fully RESTful CoAP-based interactions, allowing anything to be bound to anything. This offers a lot more flexibility than other binding solutions presented so far.

The next section describes the challenges of sensor/actuator communication which motivated us to propose this new solution. Since the solution we propose is based on the CoAP protocol and its Observe extension, we give a brief description of the protocol in section three before giving the details of the proposed solution in section four. Implementation and evaluation results are discussed in section five. Section Six discusses related work in the area and section seven concludes the paper by pointing out the way forward.

II. CHALLENGES OF SENSOR/ACTUATOR BINDING

As described above, wireless sensor and actuator nodes are used in a networked environment to achieve a specific goal. So far, different proprietary and/or open standard solutions have been applied to address the interaction of sensors and actuators. Most of the solutions use third-party devices, usually a gateway or a cloud service, to control the sensor/actuator interactions. This device or service handles the collection of sensor events and generation of actuator triggers. Given the possibility of interconnecting everything in the current Internet of Things setup, we can see several drawbacks in this solution. First, many users may need to initiate and control sensing and actuation from any device or any network. Changing the settings of a HVAC system using a smartphone over the Internet is an example where this solution fails to address. The other drawback of this solution is that the intermediate node has to be online all the time to provide the required binding functionality. If the device fails, the interaction between the sensors and actuators will be

disrupted. In this case, the intermediate device or service is a single point of failure for the whole sensor/actuator network. Moreover, in large networks this arrangement might create congestion or network delays since every packet passes through the intermediate device or service.

A possible alternative to the above solution is allowing direct interactions between sensors and actuators without the involvement of a third party watching over every interaction. Existing solutions attempted so far are too primitive and lack generality. One such solution is reprogramming the sensor and actuators every time we need new bindings. This is inflexible and may not be applicable for all use cases that we may have. A better solution that was attempted to create a direct binding is by putting the sensor and actuator in close proximity and starting a coupling procedure. This solution works for initial setup but lacks the flexibility of changing the binding thereafter. Other solutions only allow bindings between devices that have well-defined interfaces, limiting the Internet of Things vision that every device can interact with any other device.

In this paper, we propose a CoAP based flexible sensor/actuator binding solution that resolves the limitations mentioned above. The solution provided allows the realization of direct bindings between sensor and actuator nodes, thereby removing the dependence on gateways or cloud services to coordinate the interaction between these constrained nodes. In addition, interfaces for easy manipulation of bindings will make creation of and control over bindings easy and flexible. Before going into the details of the proposed solution, we will first discuss the underlying protocol, CoAP and its extension, Observe.

III. CoAP AND OBSERVE

Plugging sensor and actuator nodes directly into IP networks has long been quite a challenge. Traditional protocols are too heavy to be directly applied to such devices mainly due to the resource constraints of the nodes and the lossy nature of the network they are attached to [8]. Due to this fact, the use of web service technologies, which are very suitable to realize Machine-to-Machine or Internet of Things applications, was not possible until the introduction of CoAP by the IETF CoRE (Constrained RESTful Environments) working group. The protocol works in the same way as HTTP [9] and implements a minimal subset of REST. In fact, translation of requests and responses between the two protocols is also possible. Being a client/server system, a CoAP client sends a request to a CoAP server upon which the server responds with the appropriate information. The requests can be GET, PUT, POST and DELETE. Since TCP is a resource hungry process, CoAP uses UDP with confirmable requests to ensure proper delivery of packets. Fig 1 shows a simple GET request sent to retrieve the current state of a resource /s/t and the resulting response from the server.

There are different extensions to the CoAP protocol which are aimed at better supporting the interaction models that are typically encountered in these types of networks and at optimizing the protocol to handle additional functionalities. One such extension that is of interest for this

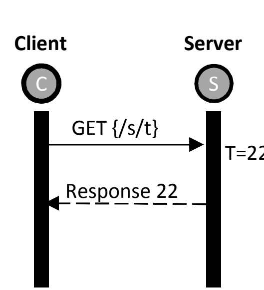

Figure 1: CoAP Operation

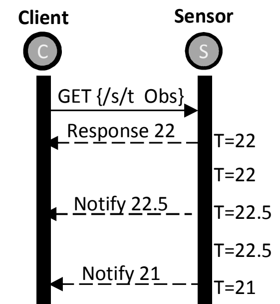

Figure 2: Observe Option

paper is the Observe functionality. The observe draft [10] states that clients may register their interest in a resource at the server once and they are notified of every state change of that particular resource. To achieve this, the client sends a normal GET request to a resource on the server by including the Observe option in the packet (Fig 2). When the server receives this message, it notifies the client of the current state and registers it as an observer so that it will be notified of new states thereafter. When it receives notifications, the client may then decide to take actions based on the significance of the change of state reported by the server. Fig 2 shows a client sending a GET request to the /s/t resource of the temperature sensor (server) by including the Observe option. The server replies with the current state and informs the client of subsequent state changes. The figure shows notification being sent when temperature changes to 22.5 and 21 respectively.

Through the Observe option, clients are guaranteed to have an up-to-date state representation of the resources on the sensor. In many real life use cases, clients need to observe the resource state to trigger some action based on the state of the resource. In practice, most changes will not be significant enough to trigger an action. This means, the client may discard some notifications if the change is not significant enough. This is not optimal for constrained devices and networks. An alternative solution to optimize this issue is conditional observation as proposed in [11]. The conditional observation draft states that upon registration, clients may specify the criteria of notification so that the server sends notifications only if the change (or the current state) meets the specified criteria. Fig 3 shows a conditional observation operation where a client sends a GET request to a server by explicitly mentioning the notification criteria (T<22). The operation is similar to the normal observe

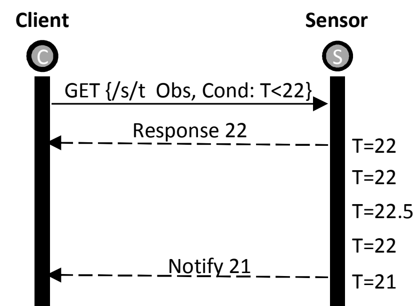

Figure 3: Conditional Observation

except the notifications. In conditional observe, even if the value changes several times, the client will not be notified unless the change results in a state that meets the criteria set by the client upon registration. Note, in the figure, that the condition option is always used along with observe. Implementation and detailed evaluation of the conditional observe option is given in [12].

IV. DIRECT BINDING

As described above, binding sensors and actuators in a flexible way is advantageous for easy deployment, independent operation and management of wireless sensor/actuator networks. Below, we will show a solution to associate a light switch (sensor) with a light bulb (actuator) using CoAP. Fig 4 shows how a traditional gateway-based solution works. The Initiator, usually the gateway, registers at the sensor to be notified about state changes of a particular resource by sending a (conditional) observe request. Whenever such an event occurs, the sensor notifies the gateway. Next, the gateway takes the initiative to trigger the actuator. The solution we propose is given in Fig 5. A simple use case we will use to demonstrate this solution is the use of a non-constrained device (e.g. smartphone) to setup a binding relationship between a light switch (sensor) and a light bulb (actuator). The resource of interest on the sensor is /gpio/btn while /lt/on is the resource of interest on the actuator. We will show how the non-constrained device may retrieve and modify the binding at a later time too. In our solution, the binding is initiated by a non-constrained device which is connected to the Internet. As shown in Fig 5, to establish the binding, the initiator sends a binding request to the sensor, the binding request expressing the resource states of interest using observe or conditional observe. To differentiate the binding request from a regular observation request, we must include binding specific information such as the actuator IP address, port number, resource of interest on the actuator, and the payload that should be applied to the actuator. To this end, we introduced four new options used to carry this binding information from the initiator to the sensor. The first option is BIND_URI_HOST. This option holds the IP address of the actuator that will be notified when events occur. BIND_URI_PORT, if present, indicates the UDP port of the

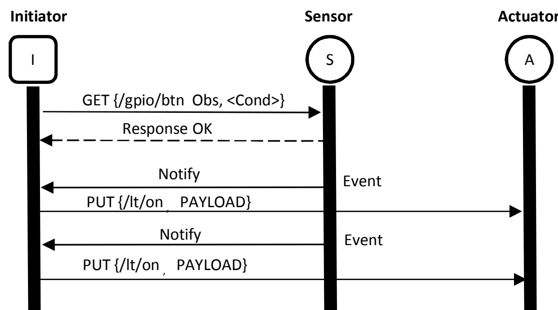

Figure 4: Gateway-based Sensor/Actuator Interaction

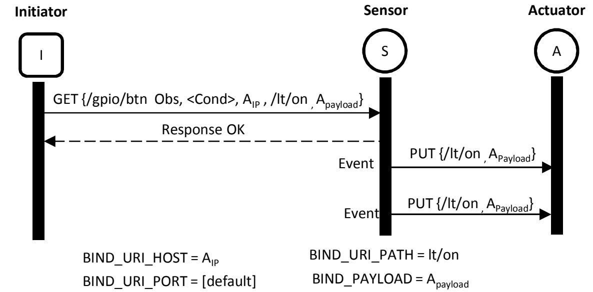

Figure 5: Direct Interaction through Binding

actuator. If not present, the default CoAP server port is assumed. The third option introduced is the BIND_URI_PATH. This option contains the path to the resource of interest on the actuator. Based on the event, the actuator has to be notified to take action. The actions are sent in the BIND_PAYLOAD option (e.g. on a lighting system 1 may mean Switch On while 0 may mean Switch Off). Currently, we assume that the CoAP method to be applied on the actuator is the PUT method. It is clear that these options contain all information needed to generate a CoAP request for the actuator, namely a PUT request to coap://[BIND_URI_HOST]:[BIND_URI_PORT]/BIND_URI_PATH, the PUT request using the content of the BIND_PAYLOAD option as its payload. If the BIND_PAYLOAD option is not present, its value is assumed to be equal to the current sensor resource state that will trigger the execution of the binding, i.e. execution of the CoAP request for the actuator.

When the sensor receives the request with these options included, it will register the actuator as an observer instead of the initiator. Whenever events occur (i.e. resource state changes that satisfy the observe or conditional observe request) the sensor notifies the actuator by sending a CoAP PUT request, using the BIND_PAYLOAD info as the payload of the message. The actuator may take different actions based on the content in the payload. It is also possible to provide observation criteria as per the conditional observe draft. As can be seen from the figure, the initiator is not involved in further notifications once the binding relationship is established.

To make the binding relationship very flexible and future management easier, we added a resource, named /binding, on the sensor so that the sensor's binding information is exposed to interested (and authorized, e.g. using DTLS) parties. This resource may then be used for future binding management (modification, deletion or addition) from any device or network. Following up on our smartphone use case, a user may establish the binding relationship using her smartphone from the Internet so that a specific light switch is associated with a specific light bulb. For executing the binding, the smartphone does not need to be online. If the same, or different, user would like to change this binding relationship at a later time, she can query the /binding resource for list of existing bindings from the sensor and send the update message to the sensor.

V. Implementation and Evaluation

We used Erbium on Contiki 2.6 to implement our proposed solution on constrained devices [13]. The non-constrained devices were programmed in CoAP++, a C++ implementation of the CoAP protocol using click router. To support the binding concept, the aforementioned four new options were added, namely BIND_URI_HOST, BIND_URI_PORT, BIND_URI_PATH, and BIND_PAYLOAD to both Erbium and CoAP++. Both the sensor and actuator nodes were Zolertia (Z1) nodes simulated in Cooja. The basic scenario we tried to simulate is the interaction between a light switch (as sensor), identified by the /gpio/btn resource, and a light bulb (as actuator), identified by /lt/on. The pressing of the switch is simulated by reading values from a random sequence of 100 0's and 1's. If there is a transition from 0 to 1 or vice versa, in subsequent readings, this indicates a button press. This will trigger a notification to be sent to the observers. Fig 6 is a Copper screenshot showing a previously created binding and Fig 7 shows a cooja sensor sending notification directly to an actuator.

To see the effect of different network topologies on the performance, we tested three different topologies. In the first topology (Fig 8a), the initiator, the sensor and the actuator nodes are all 2 hops away from each other. The second topology deliberately puts the sensor in the middle (Fig 8b) while the third topology (Fig 8c) switches the sensor and the actuator positions, putting the actuator in the middle. For all topologies the RPL routing protocol is used.

We compared the memory footprint, the transmission delay, packet size and number of packets required to complete the communication of the proposed solution against a CoAP gateway-based solution. All tests were run 10 times for each topology and the averages are taken for comparison.

A. Memory Footprint

The original Erbium code has to be modified to support binding. The modifications include defining the 4 new options, adding code to serialize and parse the options, and a mechanism to check, update and delete bindings through the /binding resource. All this requires memory space mainly in the code (text) segment and the BSS area. This can be seen, in Table 1, showing the increased memory footprint of the binding solution as compared to the gateway-based solution, whose binding logic is programmed in the non-constrained gateway. But, it is interesting to see that despite the slight increase in memory footprint, the program code can still easily fit in the memory of most constrained nodes. From this, we can conclude that, given the advantages of direct interaction of sensor and actuator nodes to make them part of the Internet of Things, the solution is viable to be applied on constrained nodes.

Figure 6: Copper Screenshot showing list of Bindings

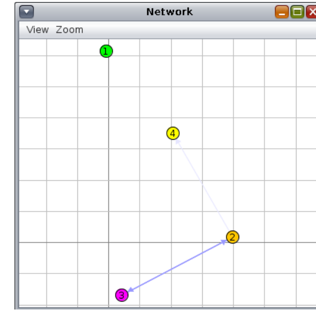

Figure 7: Sensor-Actuator Direct Notification in Cooja

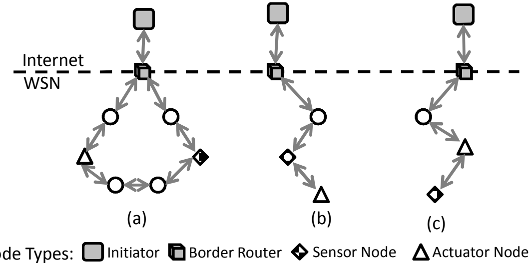

Figure 8: Topologies - (a) Two hops away (b) Sensor in the Middle (c) Actuator in the Middle

However, this doesn't come without a limitation. Being able to support an increased number of bindings leads to increased memory space requirement. The BSS section of both solutions in the table shows that when the number of observers increases, the size of the BSS increases because the program always stores a specific amount of memory for all potential observers during boot time. As memory is a very scarce resource of constrained devices, this will limit the number of observers allowed to register at a time and thus the number of bindings that can be supported. Here the gateway solution has an advantage since it may achieve scalability by aggregating multiple observe requests at the gateway avoiding one to one relationships between multiple

Table 1: Memory footprint

Max Obs	Binding Sensor				Gateway-based Sensor			
	Text	Data	BSS	Total	Text	Data	BSS	Total
0	51160	380	5894	57434	48332	362	5760	54454
1	51160	380	6160	57700	48332	362	5992	54686
2	51160	380	6426	57966	48332	362	6224	54918
3	51162	380	6692	58234	48334	362	6456	55152
4	51160	380	6958	58498	48332	362	6688	55382
5	51162	380	7224	58766	48334	362	6920	55616
6	51162	380	7490	59032	48334	362	7152	55848
7	51162	380	7756	59298	48334	362	7384	56080

actuators and a sensor.

B. Packet Size

Packet size is very important in LLNs whose MTU is very limited. In case the packets size exceeds the MTU, the packet goes through a fragmentation/defragmentation cycle until it reaches its final destination. This behavior adversely affects the performance of the network and shortens the life time of the devices which are mostly battery operated. Moreover, fragmentation comes also at the expense of an increased delay.

For most LLNs that use IEEE 802.15.4 the maximum packet size (upper layer data, upper layer headers and MAC header) must be less than 127 bytes. The packet size at the application layer can be calculated as:

Packet Size = Sizeof(CoAP-Header) + Sizeof(Token) + Sizeof(options) + Sizeof(payload)
with
Sizeof(CoAP-Header) = 4 bytes
Sizeof(Token) = 0 to 8 bytes

Sizeof(Options) differs from packet to packet, depending on the CoAP options being included in the packet. For example, Observation requests include the Observe Option which has a maximum length of 4 bytes. The Uri-Path option and payload greatly vary depending on the resource identifier and the data to be communicated. For the URI path we assume the simplified IPSO Application Framework [14] resource name for a button associated with a light switch (sensor), /gpio/btn which will be transmitted as two Uri-Path options with a total length of 9 bytes (1 byte for every option plus the length of both segments “gpio” and “btn” in the URI).

Most of these values are common for all types of communication so they do not impact the comparison of the two methods. The real difference in the two solutions can be seen at the relationship initiation packet. In case of gateway based operation the options we minimally need are Observe (1 byte) and Uri-Path (9 bytes for /gpio/btn). Including the CoAP header (4 bytes) and the token (1 bytes in this example), the total packet size will be 15. However, for direct bindings, the initial packet includes four additional options containing the information on how to trigger the actuator. Therefore, the number of additional bytes required, E_{Byte} is given by:

E_{BYTE} =
Sizeof(BIND_URI_HOST) + Sizeof(BIND_URI_PORT) + Sizeof(BIND_URI_PATH) + Sizeof(BIND_PAYLOAD)
with
Sizeof(BIND_URI_HOST) = O +16 (IPv6 address)
Sizeof(BIND_URI_PORT) = O + 2
Sizeof(BIND_URI_PATH) =
Sumof (O + sizeof(path_segment i)), with i going from 1 to # of path segments Sizeof(BIND_PAYLOAD) = O +X

In the above formula, O is the number of bytes needed for encoding the option delta and option length (between 1 and 5 bytes, but 1 in most cases). The value X depends on what we want to transmit. In our example X is equal to 1. Further, we assume the actuator uses the default CoAP server port. Using this formula, the additional number of bytes required for our example is given by E_{BYTE} = 19 + 6 + 2 = 27 Bytes. Considering the 15 common bytes, the total packet size for the binding solution will be 42 bytes..

Even if the packet size of the binding solution is bigger than that of the gateway-based solution, it doesn’t affect the network performance at all. First, this request is sent only once to establish the relationship. Once the binding is established, there is no further communication of this size. Had it been the packet size of the notification, it would, indeed, impact the network negatively. In addition, the packet size is yet in the limit of the LLNs MTU. Hence, no fragmentation will be applied that negatively affects the network performance.

C. Communication Delay

Delay is an important parameter to compare performance of different solutions. To compute the delay, we took the time difference between the occurrence of the event (at the sensor) and the reception (at the actuator) of the PUT packet. As shown in Fig 9, the delay of the gateway-based solution is always higher than that of the direct binding solution. Every notification from the sensor goes out all the way to the initiator, followed by the trigger to the actuator. As this delay depends on the number of hops that need to be traversed, this delay will be more pronounced if the network size is bigger. For our solution, the number of hops traversed depends on the routing protocol. This explains the higher delay for the first topology, as the RPL route goes via the gateway. What this means in real life use case, such as a building automation system, is that we have to wait for some time after pressing the light switch before the light is on.

In the experiments, we used X-MAC as a RDC protocol with a channel check rate of 8Hz. Nodes using X-MAC switch-off their radio communication periodically to reduce energy wastage due to passive listening. On the other hand, RDC introduces extra delay with each additional hop as the sending node must wait until the receiving node wakes up, explaining the delay values in Fig 9.

D. Number of Packets

A larger number of packets in LLNs means more power consumption at each router node and more delay. Therefore, looking at the number of packets generated by two solutions that strive to achieve the same goal is a good performance measure to compare these solutions. Since every notification goes through the gateway, the gateway-based solution creates one additional packet for every notification. If the packets are sent as confirmable requests, this number will be

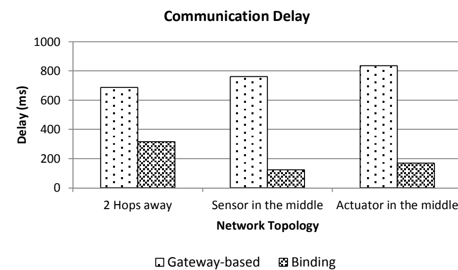

Figure 9: Notification Delay vs. Topology

doubled. As the number of sensors and actuators increases, the number of packets generated will also increase significantly. In frequently changing systems when notifications are generated frequently, the number of packets generated gets higher and higher.

VI. Related Work

There are different works that attempted to address the association of sensor and actuator nodes. An earlier work in this area was the Zigbee End Device binding. [7]. The ZigBee specification document states that devices with a similar profile can be dynamically bound by the ZigBee coordinator if they meet specific requirements such as matching cluster IDs. This solution puts a rather stringent requirement on the nodes making its flexibility quite limited. [15] proposed a mechanism to realize more flexible binding of devices in ZigBee networks. Their solution makes associations of events generated by sensors with actions of actuators irrespective of their cluster ID. However, this solution only works on ZigBee nodes and non-ZigBee devices cannot be included in the binding process. The CoRE Interfaces draft [16], also mentions the concept of bindings in the context of CoAP. Here, a binding is called the abstract relationship between two resources. The mechanism proposed in the draft allows end devices to establish a binding relationship through discovery mechanisms or through human intervention and then synchronizes the content of their resources. Three binding methods, namely polling, observe and push, are defined to achieve this synchronization. The observe method creates an observation relationship between the end points and every notification copies the content of the resource to the observer. This solution has its advantages as it provides a generic solution that can be used in interface descriptions. However, the solution focuses on synchronizing the contents of two resources on different end devices. It is not possible to execute a specific action on the other device. Additional programming logic is still required to send the appropriate trigger to the same or different actuator.

VII. Conclusion and Future work

In this paper we have shown how CoAP can be extended to support sensor/actuator bindings so that they can directly communicate with each other. In the solution we proposed, any device attached to the Internet may establish a binding between any two desired CoAP-enabled nodes and subsequently leave the network so that the two devices continue communicating with each other. As binding creation and execution is fully based on RESTful CoAP interactions, very flexible bindings between any two devices can be created. It is also possible to manage the bindings at a later time, from the external devices using CoAP messages. We have also shown that our solution does not put heavy processing and transmission burden on the constrained devices.

Taking this solution further, we try and improve the performance of these direct bindings by introducing different cross layer optimization techniques in the future. Tweaking the routing protocol to handle bindings in a special way, we may get a very good gain in further reducing the delay. Exposing existing bindings at a resource-directory-like entity, say binding directory, will also offload some tasks from the sensor and actuator nodes as far as binding relationship management is concerned. This will be another future work. Security is also another issue that we will look at in the future.

Acknowledgement

This work has been supported by VLIR Inter University Cooperation at Jimma University (IUCJU)

References

[1]. A. Z. Alkar, U. Buhur, "An Internet Based Wireless Home Automation System for Multifunctional Devices," Consumer Electronics, IEEE Transactions on (Volume:51 , Issue: 4), 2005

[2]. Vehbi C. Gungor, Gerhard P. Hancke, "Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches," IEEE Trans. on Ind. Elect., VOL. 56, NO. 10

[3]. V. C. Gungor, B. Lu, G. P. Hancke, "Opportunities and Challenges of Wireless Sensor Networks in Smart Grid," IEEE Trans. on Ind. Elect., VOL. 57, NO. 10, p. 3557, 2010

[4]. N. Kushalnagar, G. Montenegro, C. Schumacher, "RFC4919: IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs):Overview, Assumptions, Problem Statement, and Goals," IETF , August 2007.

[5]. Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), IETF, June 2013. .

[6]. I. Ishaq, et al., "IETF Standardization in the Field of the Internet of Things (IoT): A Survey," Journal of Senor Actuator Networks, 2013.

[7]. ZigBee Alliance, "ZigBee Specifications r13," 2006.

[8]. J. P. Vasseur , A. Dunkels, "Connecting Smart Objects with IP,"

[9]. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. "Hypertext Transfer Protocol - HTTP/1.1", RFC 2616, June 1999.

[10]. K. Hartke, "Observing Resources in CoAP (draft-ietf-core-observe-18)," work in progress, IETF, 2014.

[11]. L. Shi, J. Hoebeke, F. Van den Abeele, and A. Jara, "Conditional observe in CoAP (draft-li-core-conditional-observe-04)," June 2013.

[12]. G. K. Teklemariam, J. Hoebeke, I. Moerman, P. Demeester, "Facilitating the creation of IoT applications through conditional observations in CoAP," EURASIP Journal on Wireless Communications and Networking, 2013:177

[13]. M. Kovatsch, S. D, "A Low-Power CoAP for Contiki," Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2011), Valencia

[14]. Z. Shelby, C. Chauvenet, "The IPSO Application Framework (draft-ipso-app-framework-04)," IPSO Alliance , August 2012.

[15]. Y. Lee, H. Sheng Liu, M. Syan Wei, C. Peng, "A Flexible Binding Mechanism for Zigbee Sensors," 5th International Conference on, Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), pp. 273-278, 2009.

[16]. Z. Shelby, "CoRE Interfaces (draft-shelby-core-interfaces-05), (work in progress)" March 2013