electronics

MDPI

Article

Cloud Based IoT Network Virtualization for Supporting Dynamic Connectivity among Connected Devices

Israr Ullah [1,2], Shakeel Ahmad [3], Faisal Mehmood [1] and DoHyeun Kim [1,*]

[1] Computer Engineering Department, Jeju National University, Jeju 63243, Korea
[2] Department of Computer Science, Virtual University of Pakistan, Lahore 54000, Pakistan
[3] Faculty of Computing and Information Technology in Rabigh (FCITR), King Abdulaziz University, Jeddah 21577, Saudi Arabia
* Correspondence: kimdh@jejunu.ac.kr; Tel.: +82-64-754-3658

Received: 21 May 2019; Accepted: 26 June 2019; Published: 30 June 2019

Abstract: Internet of Things (IoT) is considered one of the future disruptive technologies and has attracted lots of research attention in the recent past. IoT devices are tiny sensing or actuating devices attached to daily life objects, capable of sending sensing data and receiving commands. Cloud computing technology provides tremendous computing and storage capacity over the Internet to overcome limited resources of IoT devices. Many studies are conducted on IoT device virtualization in the cloud environment to facilitate remote access and control. In the future, IoT devices will be accessed through its corresponding virtual objects. Just like the network of physical devices, there needs to be a network of virtual objects in the cyber world. In this paper, we present a concept of building a dynamic virtual network in the cloud environment among connected IoT devices. The key idea is to provide a mechanism for building a virtual network among connected IoT devices from different domains through their corresponding virtual objects in the cloud environment. This will facilitate the sharing of resources and the rapid development of diverse applications on top of the virtualization layer by establishing a dynamic end-to-end connection between IoT devices. In this study, we present a detailed design of the proposed system for building a virtual IoT network. We have also implemented three application layers protocols in OMNET++ for simulation of a virtual objects network to conduct performance analysis of the proposed IoT network virtualization.

Keywords: cloud computing; dynamic connectivity; Internet of Things (IoT); virtual IoT network

1. Introduction

Internet of Things (IoT) is designed to attach a small communicating device with everything that we want to monitor or control using the Internet. The core of IoT networks consist of two types of devices: (a) sensors that collect the context data; and (b) actuators that receive commands to control the environment. IoT is mainly focused on the connectivity of things (daily life objects with attached sensors or actuators) to the Internet so that users can remotely monitor and control a particular activity or device. IoT systems have tremendous capabilities and applications [1]. In the recent past, many giant manufacturing and development organizations have invested in this technology to realize its potential. Many projects are initiated to develop diverse IoT applications for many real-world problems and conduct research in various directions to address different aspects of IoT based systems along with associated challenges of security, authentication and authorization [2], and identification of compromised nodes [3].

To facilitate access and control of IoT devices in the cyber world, the concept of virtualization is introduced which provides a flexible interface to manipulate and interact with physical IoT devices [4].

IoT devices are represented by virtual objects in the cyber world that provide flexible connectivity between sensors and actuators in the cyber world [5]. Besides flexibility, virtualization also helps in resources' seamless duplication and sharing across diverse domains [6]. Many existing systems provide virtualization services for physical devices by creating its virtual object (VO). Later, VOs are combined to compose services that can be used to build various applications [7,8]. The virtual world is also known as the cyber world and the complete system is then referred to as a Cyber-Physical System (CPS) designed for physical world observation and control through the virtual world in IoT environments.

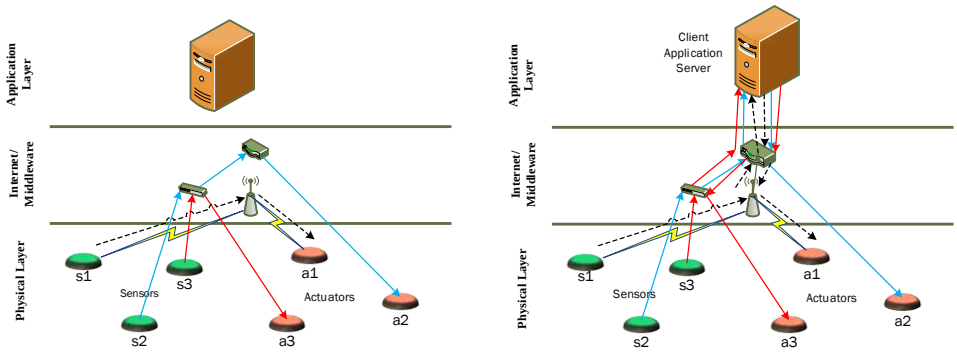
Usually, connectivity among devices is maintained in the network via routing tables. Routing tables provide an efficient and fast way for packet forwarding, but they are rigid. Making on-demand changes in routing tables is not easy, and hence the concept of Software Defined Networking (SDN) is introduced to separate network control and data plans [9]. However, SDN requires networking devices to perform specific complex operations and protocols that are not supported by constrained IoT devices. For instance, OpenFlow protocol is commonly used by SDN controllers to update flow entries in networking devices for on-demand traffic management. Several research studies can be found in the literature regarding the development of programmable SDN infrastructure for enabling the IoT ecosystem [10]. However, IoT devices are resources constrained in terms of memory and computation and provisioning support for OpenFlow like protocols will result in performance degradation.

Cloud computing technologies are aimed at provisioning stable and shared computational and storage resources to diverse users and applications [11]. With the advent of this technology, end users can have a complete computer system online as per their desired specification and budget limitation in the form of a virtual machine. In many application scenarios, virtual machines' instances launched on a cloud platform need to communicate with one another and soon virtual networks among the virtual machines will be created through virtual switches and virtual routers. After creating virtual networks in the cyber world, the functionalities of other essential networking devices such as intrusion detection devices, load balancers, firewalls, etc. were also virtualized in the form of network function virtualization (NFV). NFV framework requires three components: (a) software implementation of network function; (b) infrastructure for the deployment of virtualized network function; and (c) management and orchestration framework for virtualized network function. The conventional product development life cycle requires rigorous designing, testing, and standardization before release, which was very costly and time-consuming. NFV has brought a revolution in product development as it can quickly be deployed, tested in the real environment and easily upgraded when required.

Previously, most of the studies were focused on sensors virtualization in a cloud environment. All of the data collection and processing operations on connected virtual objects were handled in applications logic, and there was no such need for virtual network formation. However, with the advent of IoT, two types of devices are connected, i.e., sensors and actuators. The major difference between pure sensor network based and IoT based applications is that usually sensor-based applications are used for monitoring and analysis only, whereas IoT based applications are used for process automation. In IoT based applications, after careful analysis of sensing data through simple rule-based schemes or advanced machine learning algorithms, commands are sent to associated actuating devices to perform desired operations. In other words, there exists somehow a linkage or network among sensing and actuating devices. Traditionally, a configuration for the establishment of this network was either done in physical IoT devices or at the application layer as shown in Figure 1a,b, respectively.

It is very difficult to change network settings in IoT devices and we have to physically or remotely access each device and then make desired modification. With a growing number of connected IoT devices, it has become impossible to make changes in individual IoT devices. Network setting at the application layer requires full control access to corresponding IoT devices, which is normally possible when the devices and application belong to the same owner. However, this approach is not favorable for sharing the underlying physical IoT network infrastructure. After IoT devices' virtualization in a cloud environment, we can easily update and replicate virtual objects to seamlessly share a single IoT device among many dependent applications. As the virtual objects reside in the cloud infrastructure, virtual IoT

networks can therefore be established among related virtual objects in the cloud environment. This paper presents the concept of virtualized IoT network formation over the cloud environment among connected IoT devices. To the best of our knowledge, there is no study published in the literature related to virtual IoT network formation among virtualized IoT devices and hence the subject matter of this paper. Recent development in various networking and cloud infrastructure technologies enabled us to explore the concept of IoT network virtualization. Figure 2 shows the three most relevant and supporting fields that led to the development of virtualized IoT networks.



(**a**) IoT devices with preconfigured connectivity. (**b**) IoT devices connectivity established through application server.

Figure 1. Conventional approaches for establishment of IoT networks.

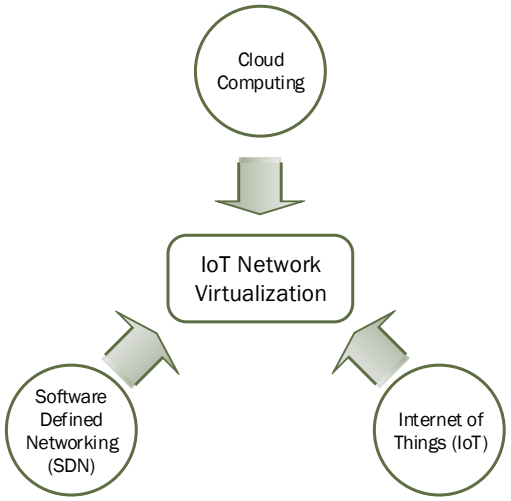


Figure 2. Convergence and evolution of IoT network virtualization.

The rest of this paper is organized as follows: a brief description of related work is covered in Section 2. The concept of IoT network virtualization is presented in Section 3 with three use-case scenarios. Section 4 presents the implementation design for IoT network virtualization through simulation in OMNET++ with a detailed description. Simulation results and discussion are given in Section 5. Finally, we conclude this paper with an outlook to our future work in Section 6.

2. Related Work

Since inception, IoT has attracted too much research attention, and it is considered among the future disruptive technologies having the potential to transform every aspect of human life. Various IoT based applications are developed for building a smart environment in various service sectors of smart cities e.g., smart homes, smart health, smart education, security, entertainment, and industry,

etc. In the past, WSN was assumed to be operated inside a particular domain, and researchers were against the use of protocol stack (due to resource-constrained devices), and a globally unique ID for each device was not essential. Advancement in information and communication technologies (ICT) and microelectronics has enabled billions of IoT devices connected to the Internet with a globally unique ID. Later, this concept of the virtual sensor network (VSN) was used to refer to a subset of sensor nodes dedicated to a specific task [12]. VSN provides logical connectivity between the selected sensor nodes, though these selected nodes may not be physically connected as shown in Figure 3. In this example scenario, there exist two virtual sensor networks (a) for temperature monitoring (b) for habitat monitoring. The sensor nodes in both networks provide logical connectivity among the sensor nodes of each network.

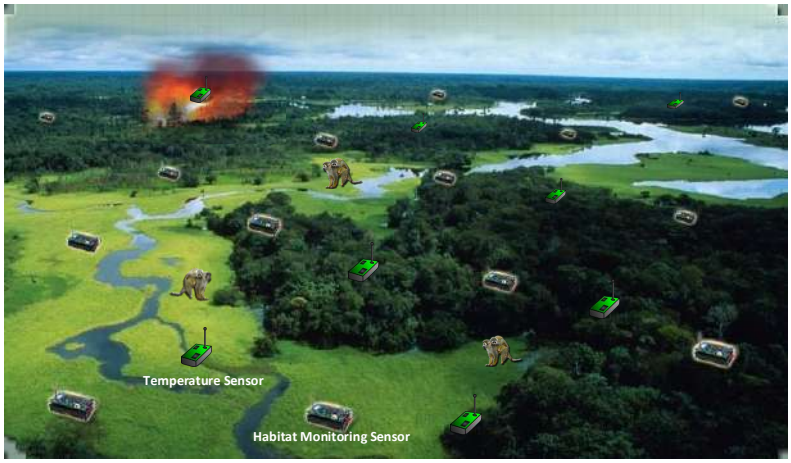


Figure 3. Example scenario of two virtual sensor networks (VSN) for temperature and habitat monitoring.

Various approaches are adapted to exploit the potential and capacity of existing physical networks. Among the initial efforts in this direction include the concept of the overlay sensor network (OSN). An overlay network is established on top of another physical network that provides the infrastructure for communication. Usually, in overlaying the networks, the underlying network is based on entirely different technologies from the one that is overlaid on top of it. Packets from one network are encapsulated inside the packets of supporting networks to enable services that are not provided by an existing network. Commonly used distributed systems such as client–server applications, peer-to-peer networks, and cloud computing are good examples of overlay networks as individual nodes in these types of networks are running on top of the worldwide Internet. The Internet itself is an overlay network originally built on top of existing telephone lines [13]. Overlay networks are also framed on top of wireless sensor networks in which a virtual topology is created on top of the physical topology in the WSN. Such overlay networks are formed by a subset of nodes in the WSN, and these nodes are not needed to be in the neighborhood of one another or have a one-hop communication link. A virtual link is assumed among the nodes of overlay network which may correspond to a complete path spanning over multiple nodes in the underlying physical network. For a seamless integration of diverse networks, the implementation for overlay network formation is done in the application layer through packets' encapsulation. However, some overlay networks may require slight modification in underlying layers of the protocol stack. [14] presents a novel access architecture for QoS provisioning in wireless sensor networks by constructing an overlay network on top of 802.11 WLAN. The overlay network acts as a control plane to ensure fairness among active communication flows with maximum bandwidth utilization.

IoT devices are supposed to be small battery powered devices having limited storage, computation and communication capabilities. To overcome resource constraint issues in sensor nodes and IoT devices, various new models are developed. Cloud-based architecture is commonly used to address

this issue. Madria et al. present a concept of sensor cloud to interconnect geographically diversified networks [4]. They have used virtual sensors to address resource constraint issues in physical sensors. A test-bed has been developed in [15] a based sensor cloud to support multi-model and multi-user sensor networks based applications. A cloud-based architecture Cloud4sens is proposed [16] to provide a remote interface for sensors monitoring and controlling. The Things Network provides a set of tools that are more focused on rapid IoT application development with minimal/zero coding efforts through drag and drop operations [17]. Efficient resource allocation in the virtual sensor network environment is a challenging problem, and various solutions are proposed in the literature to address this issue [18–21]. The concept of virtualization in WSN is not new [22–24]. Bhattacharya et al. present the idea of community sensor grids for virtualization and sharing of sensing resources across the domains [6]. Several studies present the prototype implementation of sensor virtualizations, e.g., Raveendranathan et al. have developed a model for sensor virtualization in body sensor networks [25]. They have also implemented the proposed model using the SPINE2 architecture, which is an open source framework to support domain-specific signal processing and sensing operation of wireless sensor networks. Levis et al. have developed a tiny virtual machine Mate for sensor networks [26]. Mate provides interfaces to encode complex programs in a very small size (less than 100 bytes), thus avoiding transmission and storage issues in sensor networks. Leontiadis et al. have developed a platform SenShare to decouple sensor networks infrastructure from applications running on top of it [27]. Thus, multiple applications can share the same underlying infrastructure to perform different operations. Furthermore, virtualization in sensor networks can help in the separation of concerns in conventional WSNs. Previously, WSNs were managed and owned by a single entity and the same entity was responsible for sharing their network for service provisioning to various client applications. In the future, a logical network can be established among the virtualization servers for different domains to support resource sharing for provisioning coordinated services to client applications for the future smart world as shown in Figure 4. For every domain, we may have a virtualization server that holds information about the associated IoT devices in the form of virtual sensors or virtual objects. These virtualization servers can be connected to share resources information across the domains.

Most of the existing studies are focused on the virtualization of IoT devices in the cloud environment so that it easily be replicated and shared among the different applications [28,29]. A study presented in [30] can be considered among the earliest attempts of IoT devices virtualization, although the authors did not use this term explicitly. However, cloud computing technologies provide centralized, reliable access to a high volume of data along with tremendous computation power, but it is not suitable for delay sensitive applications due to significant communication delay in sending data to the centralized cloud server for processing. To address this challenge, edge computing technologies are gaining attention where computations are pushed closer to the origin of data to avoid unnecessary communication delays [31,32]. More recently, numerous studies can be found in the literature regarding virtualization of IoT devices, and interested readers can find comprehensive surveys on virtualization of wireless sensor networks in [33,34]. SDN based virtualization solutions for IoT devices [35–37] are focused on remote management of IoT devices to make necessary configuration changes, but this approach requires computation and storage resources that may result in performance degradation. Cloud-based service virtualization provides a centralized and shared repository of virtual objects for registered IoT devices so that they can easily be shared and replicated. However, all the data collection and processing operations on connected virtual objects were handled in applications logic, and there was no as such need of virtual network formation. With the advent of IoT, two types of devices are connected i.e., sensors and actuators [38]. In IoT based applications, there exist somehow a linkage or network among sensing and actuating devices that can be realized in the form of virtual IoT formation in the cloud environment among related virtual IoT devices. To the best of our knowledge, there is no study published in the literature related to virtual IoT network formation among virtualized IoT devices and hence the subject matter of this paper.

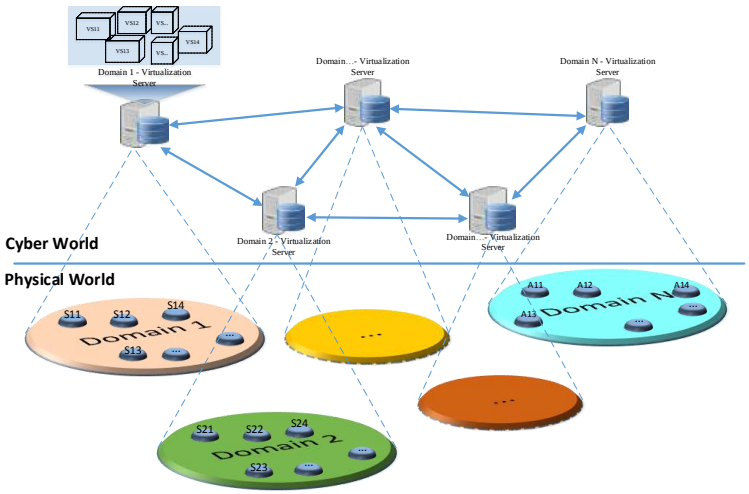


Figure 4. Network of virtualization servers for building shared infrastructure across the domains.

3. IoT Network Virtualization (INV)

The idea of IoT network virtualization is borrowed from the concept of NFV in the context of SDN and cloud computing. NFV enables the rapid development of the functionalities of real devices in the form of a software package and its convenient deployment in the cloud environment. Moreover, through NFV, these virtual devices can easily be updated, upgraded, replicated and shared.

IoT allows the connectivity of small sensing and communicating devices to the Internet and are attached to daily life objects for remote monitoring and control. Billions of IoT devices are expected to be connected to the Internet in the near future, which will be generating an enormous amount of data. Many research studies suggest and prefer the connectivity of IoT devices to a cloud environment for convenient storage and processing of collected data using advanced data analytics and big data processing schemes supported by the cloud platform. Leading cloud service providers such as Google, Amazon, Microsoft, etc. also provide support and APIs for connectivity of IoT devices directly to their platforms so that its data can easily be consumed and processed by respective client applications. Like virtual machines and virtualized network functions, we can have a virtual representation of IoT devices in the cloud environment commonly referred to as virtual objects. The major difference between NFV and virtual objects is that NFVs have no physical device backing them, whereas virtual objects are backed by real IoT devices. Once IoT devices are connected, and corresponding virtual objects are created in the cloud environment, then a virtual IoT network can be established among associated virtual objects.

Figure 5 shows the conceptual view of virtual IoT network formation among virtual IoT devices (virtual objects). Figure 5 also presents a brief summary of the evolution in network types. A type-1 network corresponds to the conventional physical networks where routing information is maintained at intermediate routers. For any changes in the network setting, each router needs to be accessed individually. When the network size grows, individual maintenance of networking devices becomes a tedious job. To overcome these issues, networks of Type-2 were introduced, i.e., SDN. In SDN, data and control plans are separated. The network administrator can use centralized controllers for network configuration and maintenance. The Controller interacts with individual networking devices to reflect necessary changes in their internal routing tables. Networking devices do just the data forwarding as per the rules defined by the controller in their routing tables. The concept of virtualized IoT networks can be considered as the Type-3 networks where the network among connected IoT devices is maintained in the cloud environment through corresponding virtual objects.

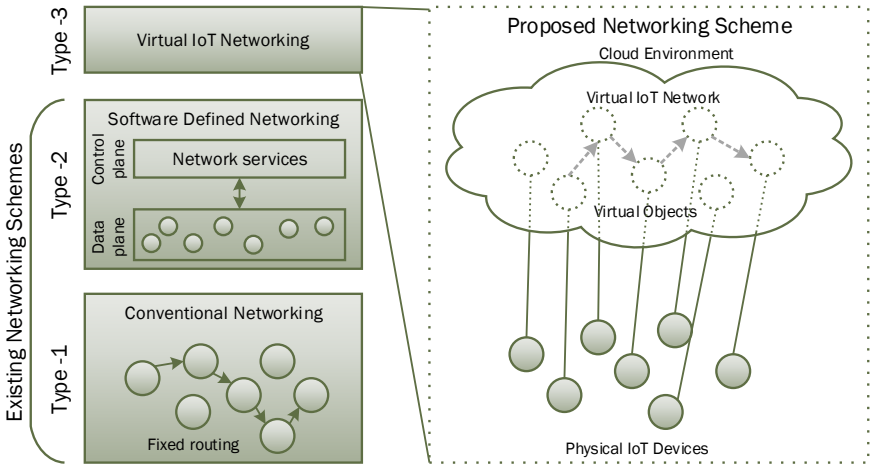


Figure 5. Conceptual view of a virtualized IoT network.

Figure 6 shows a more detailed view of virtualized IoT network formation for diverse applications in the cloud environment. In this proposed architecture, client applications only have to communicate with the centralized controller to specify their required network configuration. Just like a controller in SDN networks, the controller module is centralized, responsible for network formation and maintenance. Figure 6 shows three virtualized IoT networks setup on-demand for different client applications. The proposed architecture supports sharing of the same virtual objects across different networks.

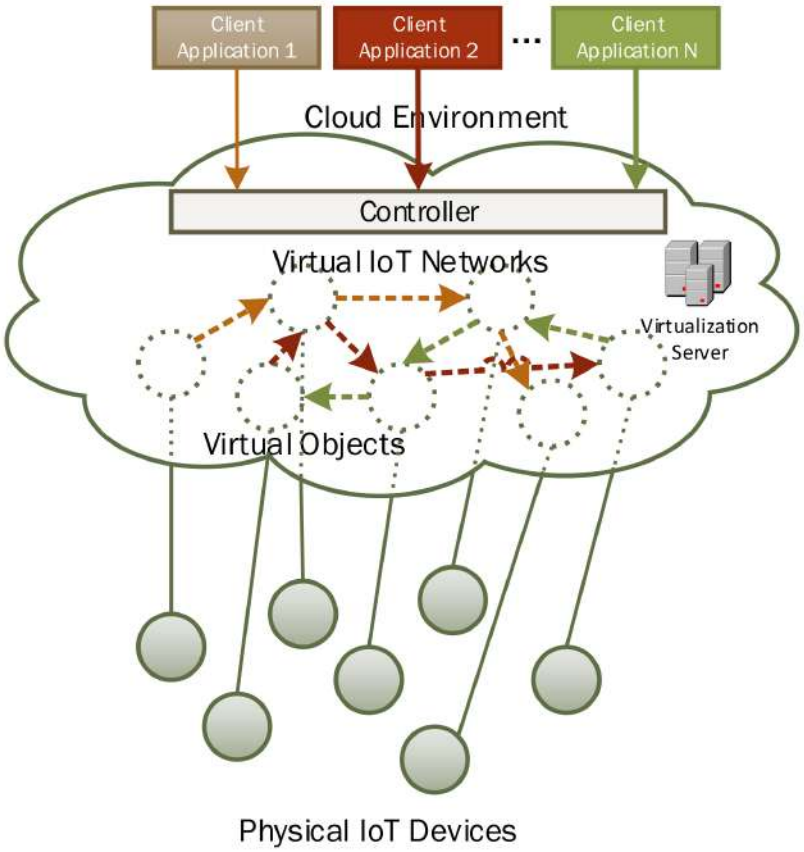


Figure 6. Virtualized network formation for diverse applications in the proposed IoT network virtualization.

Layering architecture of the proposed system for IoT network virtualization is given in Figure 7. It has three layers: (a) the Physical layer consists of actual IoT devices that can be sensing and actuating devices connected to the system. The IoT devices can be distributed across different domains sharing the same goals and objectives; (b) the virtualization layer acts as a middleware between client applications and physical IoT devices. Virtual objects are created in this layer for every connected IoT device, and these virtual objects are used by and shared among different client applications. The virtualized IoT network is established among related IoT devices as per client application demand and desired settings. Virtual objects provide an interface to the actual IoT device to consume its services across different applications. (c) Various applications are hosted in the application layer for consuming the services of the underlying virtualization layer. Applications from any domain can have access to the shared resources through the virtualization layer. Figure 7 also shows the virtualized IoT network formation in the virtualization layer for two different clients in application layers. Applications express the required number of IoT devices with desired connectivity settings through a dedicated interface and the logic is reflected in the form of a virtual network among related virtual objects.

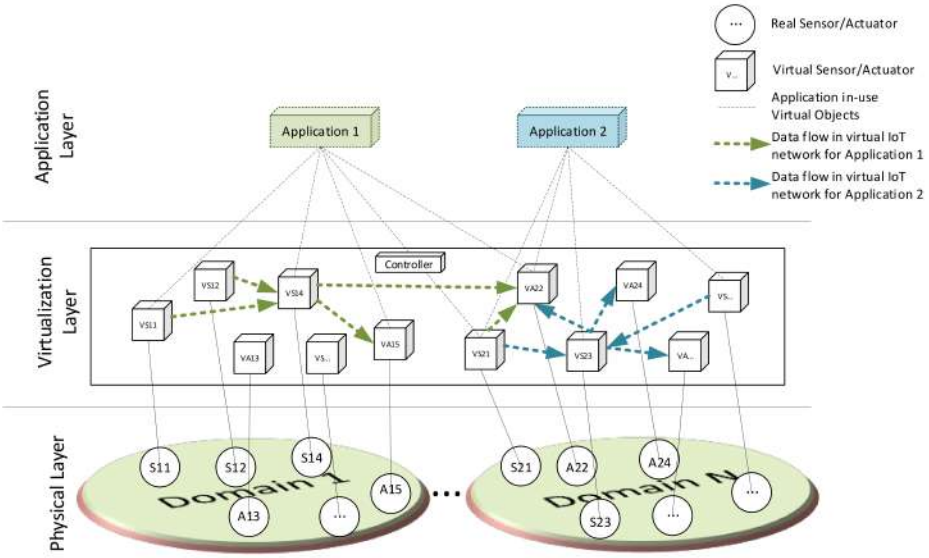


Figure 7. Layered architecture of proposed IoT network virtualization.

The essential components of the proposed system for IoT network virtualization and its functioning is shown in Figure 8 via a sequence diagram. The process is initiated with IoT device registration where every IoT device posts its profile information to a pre-configured virtualization server by sending a registration request. A typical virtual object profile for an IP camera has information as shown in XML format in Figure 9. After the necessary validation of device profile information, a virtual object is created for a corresponding IoT device and an acknowledgment message is sent to the respective device. The virtual object is used by a virtualization server for onward communication with a corresponding IoT device. Afterwards, the user initiates a request via client application for the desired type and number of IoT devices. The Controller processes the request and retrieves virtual objects' profiles of matching IoT devices as per user specified criteria. Upon receiving virtual objects' lists at client application, the user specifies and configures desired settings. Afterwards, user desired settings are deployed via client application and the request is submitted to the controller. The Controller is responsible for establishing desired network settings by manipulating virtual objects in the virtualization server. Dynamic connections are made among related virtual objects as per user desired settings by updating the mapping list. Afterwards, an activation command is sent to corresponding IoT sensors to initiate data transmission. Sensing data are received by corresponding virtual objects and then the actuation command is forwarded to the next connected virtual object after selecting the mapped device from the mapping list. The actuating device performed the desired

operation and acknowledgment message is sent to the client application. The process continues until the device activation time is over. Connectivity information for a particular virtualized IoT network is purged from mapping list when its activation time is over.

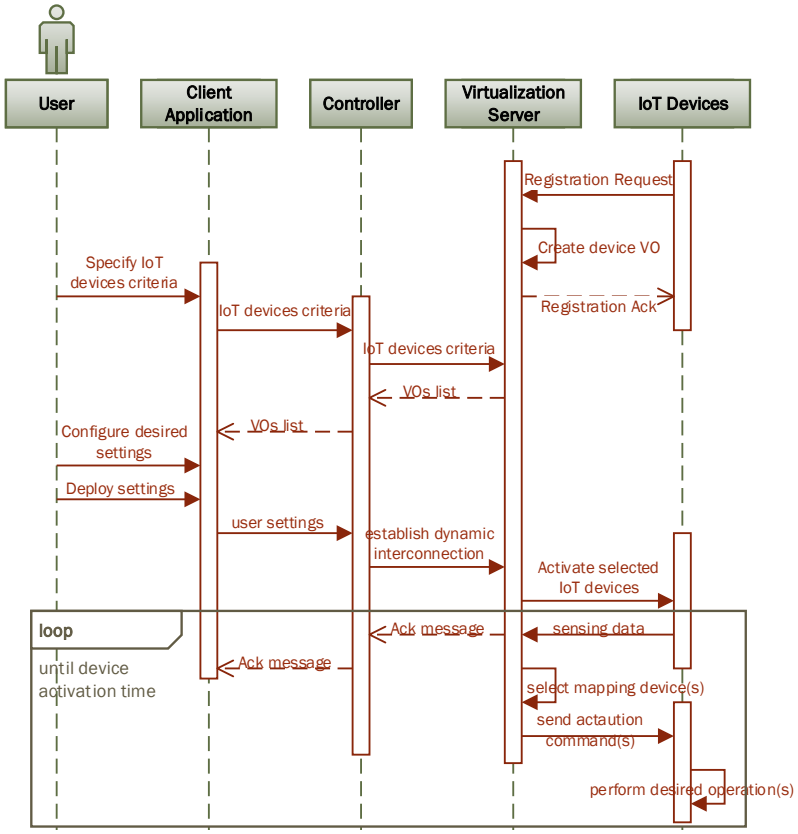


Figure 8. Operational sequence diagram of IoT network virtualization, formation and operation.

```
<?xml version="1.0" encoding="UTF-8"?>
- <Devices>
  - <Device>
      <VO_ID>16</VO_ID>
      <Uri>coap://10.102.42.125</Uri>
      <Type>IPCamera</Type>
      <Location>Room123</Location>
      <LocLat>33.455</LocLat>
      <LocLong>126.74</LocLong>
    - <Properties>
        <P>GetStream</P>
        <P>SaveStream</P>
        <P>MoveUp</P>
        <P>MoveDown</P>
        <P>MoveLeft</P>
        <P>MoveRight</P>
      </Properties>
    </Device>
</Devices>
```

Figure 9. Typical IP camera profile (XML view).

3.1. Master Mapping Table and Flow Tables

In the proposed setup, the virtualization server stores the virtual objects of all connected IoT devices. The Controller can be hosted on the same server, or deployed on a separate server if needed. Virtualization server provides virtual representation for registered IoT devices in the form of virtual objects, whereas the controller module interacts with clients to manipulate connectivity among virtual objects as desired by clients. Table 1 is the master table maintained at the controller module containing all information about the virtual IoT networks established in the cloud for a typical scenario given in Figure 7. Two applications are using the virtual objects in the virtualization layer and the

communication flow for each virtual IoT network is indicated with a different color. As shown in Figure 7, some virtual objects are shared by the two applications e.g., VS21 and VA22 are being used by both applications. Two entries are done for VS21 in the master mapping Table 1 (i.e., Entry ID 1010 and 1011) to handle and generate packets for both applications.

Table 1. Controller master mapping table with sample entries for general scenario given in Figure 7.

Entry ID	Network ID	VO ID	Source ID	Target ID	Received From	Set Source ID	Set Target ID	Send To	Entry Type	Expiry Time
1001	vIoTNet-1	VS11	-	-	S11	S11	A22	VS14	Event	11 October 2018 14:00
1002	vIoTNet-1	VS14	S12	A15	VS12	-	-	VA15	Event	11 October 2018 14:00
…	…	…	…	…	…	…	…	…	…	…
1010	vIoTNet-1	VS21	-	-	S21	S21	A22	VA22	Event	11 October 2018 14:00
1011	vIoTNet-2	VS21	-	-	S21	S21	A24	VS23	Periodic	15 October 2018 05:00
1020	vIoTNet-2	VS23	S21	A24	VS21	-	-	VA24	Periodic	15 October 2018 05:00
1021	vIoTNet-2	VA24	S21	A24	VS23	-	-	A24	Periodic	15 October 2018 05:00
…	…	…	…	…	…	…	…	…	…	…

Each entry is uniquely identified with an ID, i.e., entry ID. Every entry in the master table represents the linkage between two real/virtual objects and will belong to a particular virtual IoT network having a unique network ID. This table contains master information about the connectivity among the virtual objects and connectivity information of an individual virtual object is identified with the field 'VO ID'. Every packet generated in this virtual IoT network architecture may be destined for a single/multiple target nodes. Source and Target ID indicate the origin and destination for a particular communication flow. When a virtual object receives a packet from a real device, then it sets the source and target ID as per rules specified by the master mapping table. The field "received from" is used to identify the previous hop of the packet so that it can forward the packet to the next node as identified by the "sent to" field. The entry type field indicates the type of data flow, e.g., event-based, periodic, etc. Expiry time indicates the validity time of a virtual IoT network and all the entries will be purged when its validity time expires.

Sample entries in the flow table for VS21 shown in Figure 7 are given in Figure 10. This virtual object is shared by the two applications. A separate entry is made for each application to specify flow rules for data forwarding at the virtual object level. Let us suppose that application 1 builds an event-based system, and its rule will be triggered if the sensor data value is greater then a certain threshold. For a second application, we assume a periodic control system that sends a command to actuator after a regular interval of 0.1 s in this example. When the data packet is received at the virtual object from the real sensor, then its source and target ID fields are set. A single packet received at the virtual object can be forwarded to multiple target nodes by having multiple entries in the flow table. Furthermore, if the flow entry is of an event type, then the condition field is checked if true and then a corresponding action is taken. If flow entry type is periodic, then the interval field is used to set the timer for the next packet transmission.

Entry ID	Network ID	Source ID	Target ID	Received From	Entry Type	Condition	Action	Interval	Set Source ID	Set Target ID	Send To	Expiry Time
101	vIoTNet-1	-	-	S21	Event	If data > threshold1	Send message (command= Action1)	-	S21	A22	VA22	11/10/2018 14:00
102	vIoTNet-2	-	-	S21	Periodic	-	Send message (command= Action2)	0.1	S21	A24	VS23	15/10/2018 05:00

Figure 10. Virtual sensor VS21 flow table with sample entries for a general scenario given in Figure 7.

3.2. Use Case Scenarios

In the following sub-sections, we present three different use case scenarios to illustrate the working and utility of the proposed IoT network virtualization concept. In each scenario, we present a particular application scenario that requires some specific type of connectivity among related IoT devices. Through these examples, we will explain how a virtualized IoT network is established among corresponding virtual objects to achieve the desired application objective.

3.2.1. Scenario 1

In this example, we consider a simple scenario where a user wants to develop an IoT based automatic door opening application in a small living area that consists of three rooms. For simplification, we consider two types of IoT devices in this network, i.e., motion sensor and door actuator as given in Figure 11. We have a motion sensor deployed at the door of every room and door opening and closing is controlled by a corresponding actuator. A total of six virtual objects need to be created at the virtualization layer. This scenario requires a direct one-to-one connection between IoT devices through virtual objects based on simple rules using condition/action. A user can get a virtual list through a client application interface and specify desired connectivity among corresponding sensor and actuator. Users can also specify simple rules indicating a sensitivity level of motion sensing to activation of a door opening actuator. After deployment, the controller will update settings of each virtual object to establish desired connectivity settings among related virtual objects and activation commands will be sent to motion sensors. The motion sensor will start sending motion sensing data at a designated interval to a corresponding virtual object where simple rules will be applied to determine whether or not to send activation commands to door opening actuators.

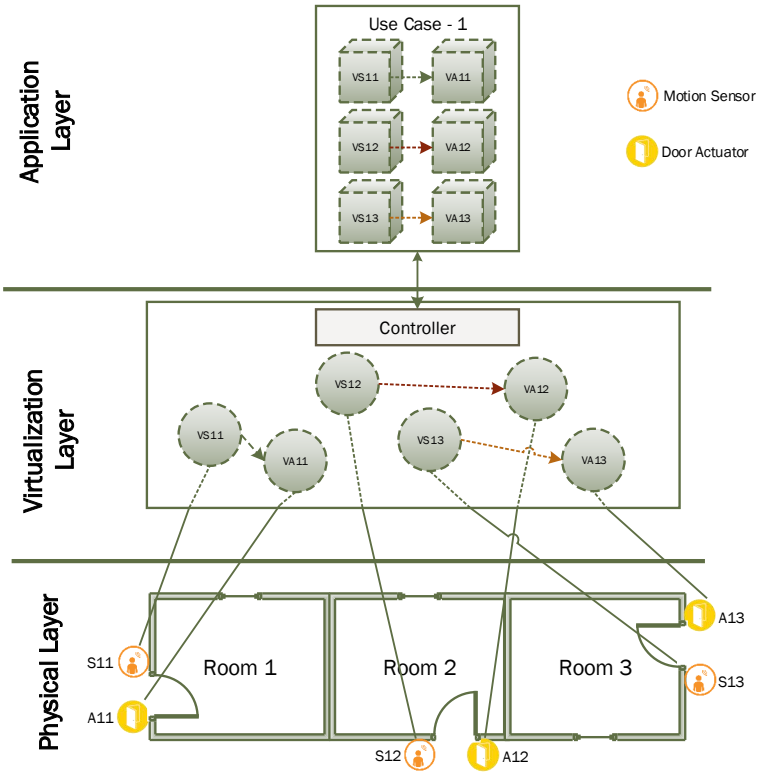


Figure 11. An application scenario for virtualized IoT network formation having one-to-one device connectivity.

3.2.2. Scenario 2

In this case, we consider an example scenario where the client wants to develop an IoT based fire safety application in a small living area that consists of three rooms. For simplification, we consider three types of IoT devices in this network i.e., smoke sensor for fire detection and two actuators i.e., alarm and sprinkler. We have a smoke sensor and sprinkler deployed inside every room and alarm is installed in the corridor as shown in Figure 12. For this scenario, a total of seven virtual objects are needed to be created at the virtualization layer. The user can get a virtual objects list through a client application interface and specify desired connectivity among corresponding sensors and actuators. Users can also specify simple rules indicating the sensitivity level of smoke sensing for activation of a corresponding sprinkler and alarm. After deployment, the controller will update settings of each virtual object to establish desired connectivity settings among related virtual objects as shown in Figure 12 and activation commands will be sent to smoke sensors. The smoke sensor will start sending smoke sensing data at the designated interval to a corresponding smoke sensor virtual object where simple rules will be applied to determine whether or not to send activation command to sprinkler and alarm actuator. This scenario requires complex interconnection between IoT devices through virtual objects by establishing one-to-one and many-to-one interconnections. One-to-one connectivity is established between the smoke sensor and corresponding sprinkler, whereas all smoke sensors are also connected to a single alarm actuator which requires a many-to-one type of connectivity. Application logic for this scenario is expressed in the application layer where its implementation is realized in the virtualization layer through the establishment of a virtualized IoT network among related virtual objects.

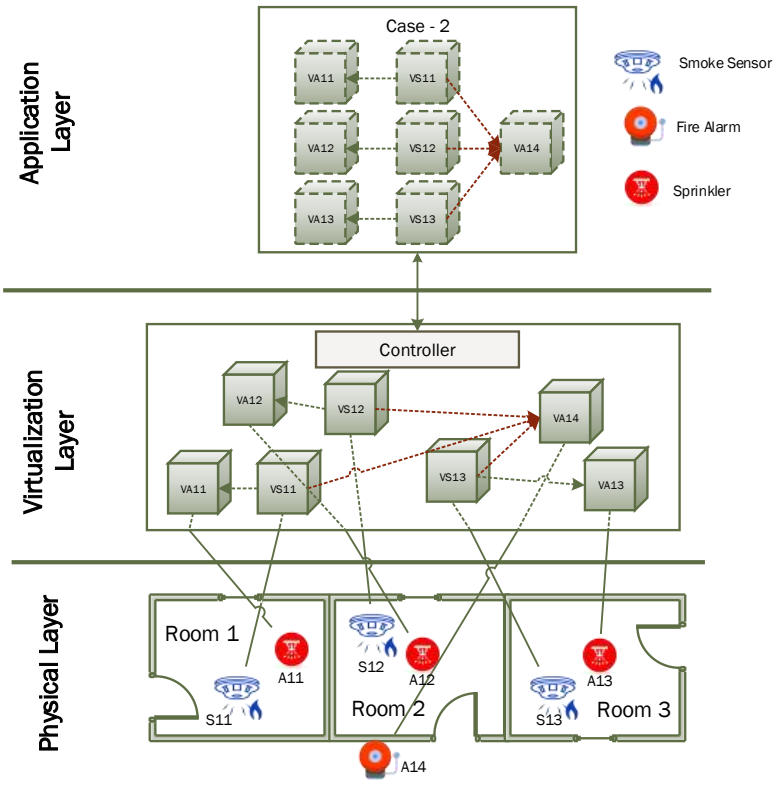


Figure 12. An application scenario for virtualized IoT hybrid network formation having one-to-one and many-to-one device connectivity.

3.2.3. Scenario 3

This scenario requires complex interconnection between IoT devices through virtual objects by performing various data fusion, integration and aggregation operations. This scenario is different from

the other two cases as here certain virtual objects have the capacity to both receive and send the data. An example application scenario is given in Figure 13 to better explain this scenario. In this case, the client wants to develop an IoT based indoor environment application in a small living area that consists of a big hall. The user wants to control indoor temperature, lighting and air quality using this smart application. For simplification, we consider six different types of IoT devices in this network, i.e., three sensors—temperature sensor, illumination sensor and CO_2 sensors, and three actuators i.e., AC/heater, lights, and two fans. The user needs to specify the desired range for indoor temperature, illumination, and air quality. AC will be operated if the indoor temperature is above the desired range to cool down the indoor environment and the heater will be operated if the indoor temperature is below the desired range to heat the indoor environment. Similarly, if the indoor illumination level is below the user desired range, then electric bulbs will be turned on. Likewise, if indoor air quality is getting worse (more CO_2 concentration than specified level), then the fan will be operated to maintain air quality.

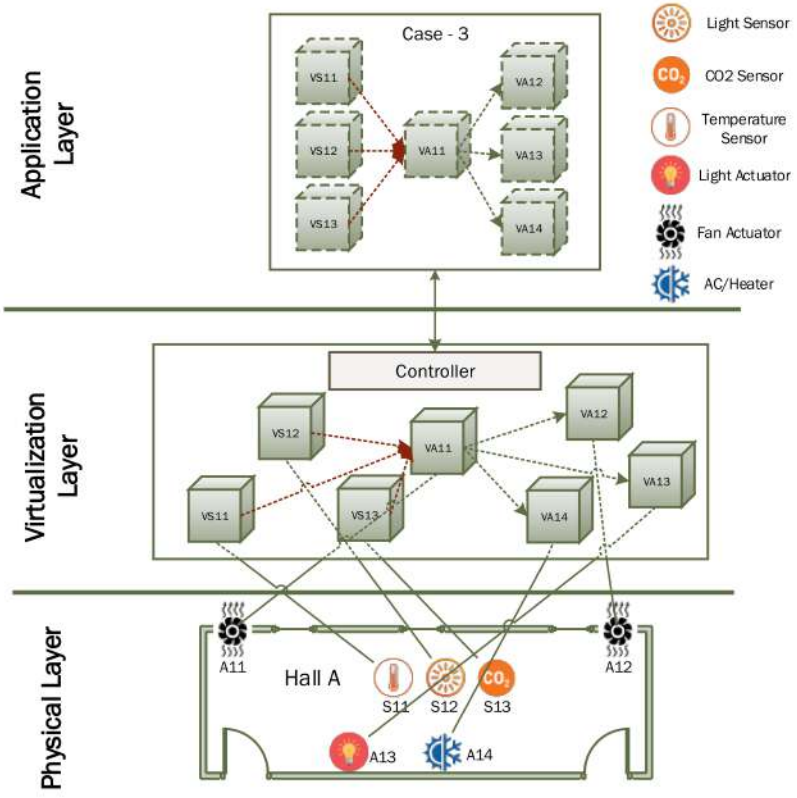


Figure 13. An application scenario for virtualized IoT complex network formation.

For this scenario, a total of seven virtual objects are needed to be created at the virtualization layer as shown in Figure 13. The user can get a virtual objects list through client application interface and specify connectivity among corresponding sensors and actuators to achieve the desired objective. The user can also specify simple rules as per the desired range for each indoor parameter for activation of corresponding actuating devices. After deployment, the controller will update settings of each virtual object to establish desired connectivity settings among related virtual objects as shown in Figure 13 and activation commands will be sent to sensors. Sensors will start sending sensing data about the indoor environment at the designated interval to the corresponding sensor virtual object. As per virtual IoT configuration, all sensing virtual objects will forward the data to a designated fan virtual sensor where simple rules will be applied to determine whether or not to send activation command to each actuator. This scenario requires the establishment of many-to-one and one-to-many connectivity between sensors and actuator virtual objects. Application logic for this scenario is expressed in

the application layer where its implementation is realized in the virtualization layer through the establishment of a virtualized IoT network among related virtual objects.

4. Implementation Design

We have configured a hybrid network simulation setup in OMNeT++ for virtual objects network simulation as shown in Figure 14. OMNeT++ stands for Objective Modular Network Testbed in C++ and it is a very popular simulator for discrete event simulation [39]. We can also perform network simulation in OMNeT++ by using an INET framework (open-source). This package includes different protocols for communication networks (wired, wireless and mobile networks) [40]. In this paper, we have used OMNeT++ 5.2 with the INET framework version 3.6.

We have developed three application layer protocols in OMNeT++ for Client, Server and IoT devices in our network. The directory structure of these protocols in the OMNeT++ environment is shown in Figure 14. The server system has virtualization application protocol *VirtualizationServerApp*, IoT device has *IoTDeviceApp* protocol at the application layer. Afterwards, we have developed an application layer protocol for the client *IoTClientApp* to send a request to a virtual objects server for configuration of the desired topology among registered virtual objects to perform the desired operation.

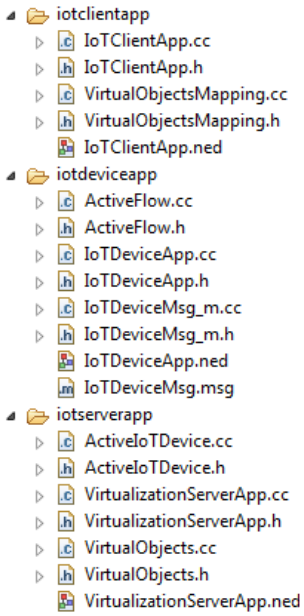


Figure 14. Directory structure of the three protocols in OMNeT++.

4.1. IoT Message Structure

Before presenting the detailed internal structure of these protocols, we first discuss the message structure that is used in these experiments. Table 2 presents a brief description of various message fields. Each message has a unique ID for its identification in the simulation environment. In these experiments, various types of messages are generated, e.g., registration request message, acknowledgment message, command message, etc. A message type is an integer number indicating the type of message and Table 3 presents a brief description of the various types of messages used in these experiments. The data field in the message structure is the most important part of the message structure. Depending upon the message type, associated message content and information are encoded in this field. The parser is used to retrieve information from the data at the receiver side. Data field size is variable as a different type of messages requires different information. The total size of the message is mainly dependent upon the size of this data field. Next, we present a brief description of the three application layer protocols used in this study.

Table 2. IoT message structure.

Message Field	Description
messageID	Unique ID of message for its identification
type	Indicate the type of message (various message types are given in Table 2)
from	Address of the originator/source of the message
to	Address of the target/destination of the message
sendingTime	The time stamp at which the message was sent/generated from source.
data	Contains the actual content of message in encoded format.
messageLength	The total size (bytes) of message includes header and data contents.

Table 3. Description of various types of IoT messages.

Message Type	Description
1	Registration request message from IoT device
2	Registration acknowledgment message from server
3	Command message received from server
4	Sensing data message from IoT device
5	Command data packet from client to actuator IoT device
10	Data request message from client end
11	Acknowledgement message to data request packet from server to client device

4.2. IoTDeviceApp Protocol

IoTDeviceApp is an application layer protocol designed for both sensing and actuating IoT devices. The directory structure of this protocol given in Figure 14 shows the source files used in its implementation. IoTDeviceMsg.msg is the message structure file that is automatically compiled by OMNet++ message compiler. This message structure is shared across all other protocols. This protocol maintains the necessary information in network description (NED) language as required by the OMNeT++ simulator. All application logic of this protocol is implemented in C++ language. This protocol also keeps track of active flows by holding information about the target device, packet sending interval and flow expiry time. The working flow diagram of this protocol is given in Figure 15.

At simulation startup, IoT nodes perform necessary configuration settings and then sets a timer for sending registration request. When the registration timer gets expired, OMNeT++ sends a timer message to the corresponding module as a self message. IoT device generates registration request message (Type = 1) by encoding its profile information in its data field and then sends this message to the underlying transport layer protocol for onward transmission to the virtualization server. The timer is set again to re-transmit the registration request if no acknowledgment is received within a certain amount of time. Upon reception of an external message, first, we check if it is a self message (timers) or an external message and then it is processed accordingly. When a registration acknowledgment message (Type = 2) is received, then device updates its internal settings as registered and cancels the previously set timer to avoid the re-transmission of registration request.

Afterwards, the IoT device may receive an external command message (Type = 3) from the virtualization server to initiate data transmission at a specified sending rate. The message is parsed and active flow is generated and its expiry timer (flow timer) is set. Then, the first data packets (Type = 4) are sent and the sensing timer is set to continuously send packets. If the IoT device is an actuating device, then it may receive a control message (Type = 5) from a virtualization server to perform a designated operation.

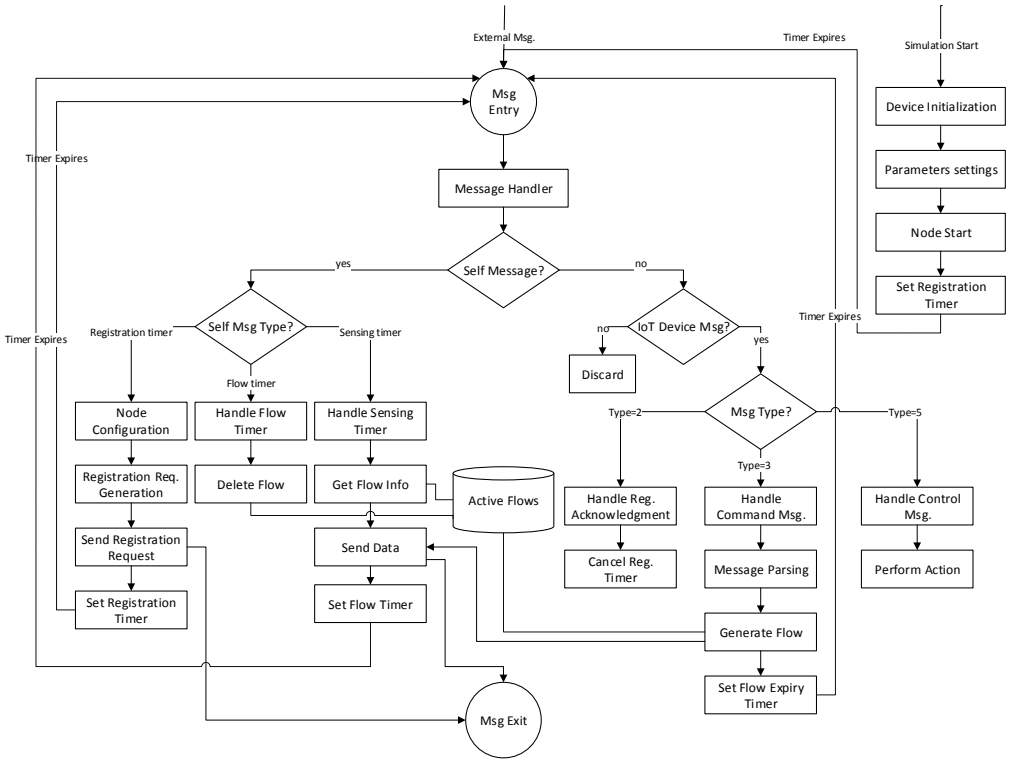


Figure 15. Working flow diagram of IoT device protocol for virtual IoT network simulation.

4.3. VirtualizationServerApp Protocol

VirtualizationServerApp is an application layer protocol designed for IoT virtualization server node. The directory structure of this protocol given in Figure 14 shows the source files used in its implementation. Application logic of this protocol is implemented in C++ language, whereas information about the active IoT devices is stored in a data structure. Active IoT devices are those devices that are currently used in some virtualized IoT network and have active communication flows. Profile information of a registered IoT device is also maintained. The working flow diagram of this protocol is given in Figure 16. At simulation startup, the virtualization server performs necessary configuration settings and then waits for an external message either from IoT devices or clients. Three types of messages are expected at the virtualization server (a) Registration request message (Type = 1), which results in the creation of virtual objects for corresponding IoT devices after necessary validation followed by sending an acknowledgment message; (b) a request message from client application (Type = 10) for the establishment of a virtualized IoT network through selection of desired virtual objects. The activation command is sent to the selected IoT device to initiate data transmission. Acknowledgment is sent to a client application; (c) sensing message (Type = 4) from activated IoT devices for onward forwarding to the corresponding target IoT device after verification.

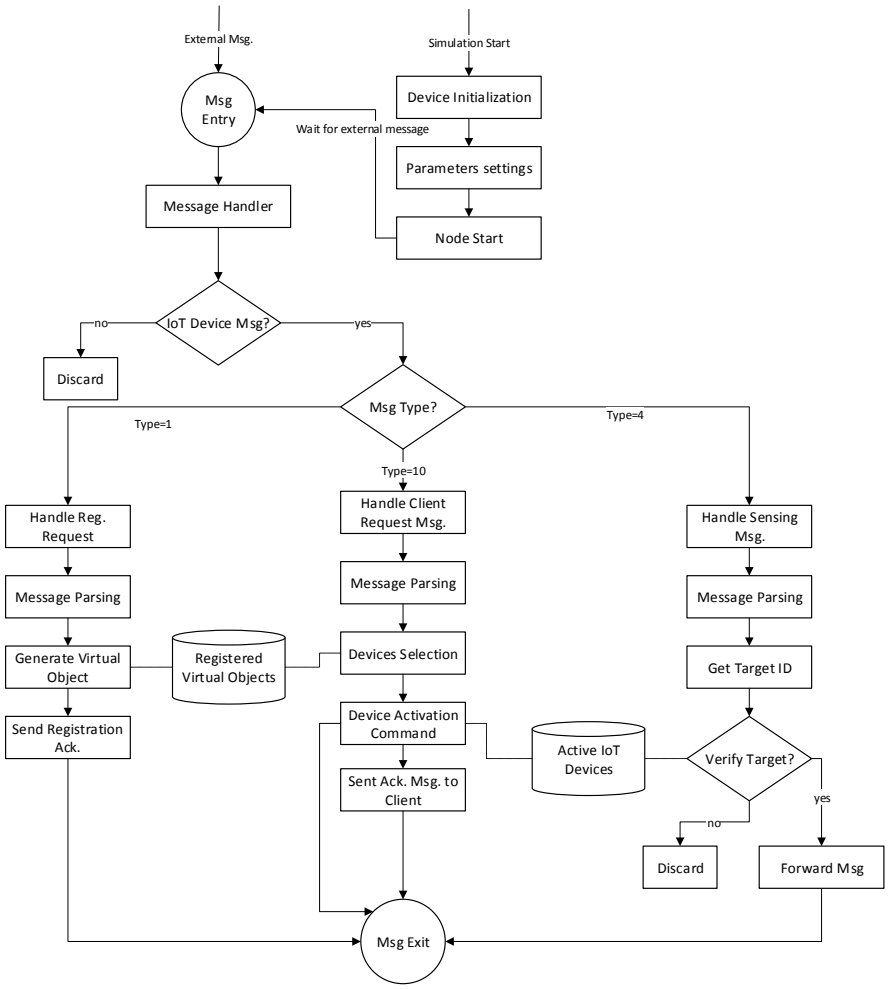


Figure 16. Working flow diagram of virtualization server protocol for virtual IoT network simulation.

4.4. IoTClientApp Protocol

IoTClientApp is an application layer protocol designed for IoT client devices. This protocol allows the user to get the desired virtual objects list from the virtualization server and specifies required network topology settings by mapping corresponding virtual objects.

The working flow diagram of this protocol is given in Figure 17. Before the simulation start-up, we specify the request sending time for the client device in a simulation initialization script file i.e., omentpp.ini. At simulation startup, the client device performs necessary configuration settings and then sets a timer for sending virtualized IoT network formation request (Type = 10). When the request timer gets expired, OMNeT++ sends a timer message to the corresponding module as a self message. The client device generates a request message (Type = 10) by encoding its desired virtualized network topology information in its data field and then sends this message to the underlying transport layer protocol for onward transmission to the virtualization server. The request timer is set for its re-transmission if no acknowledgment is received from the server within the specified time. After the reception of the acknowledgment message, the client device stores requested mapping information in its database. A mapping timer is also set to purge the corresponding mapping entries from its data store when it is no longer needed. A copy of the sensing and control message is also sent to the client device by the virtualization server to keep its status updated.

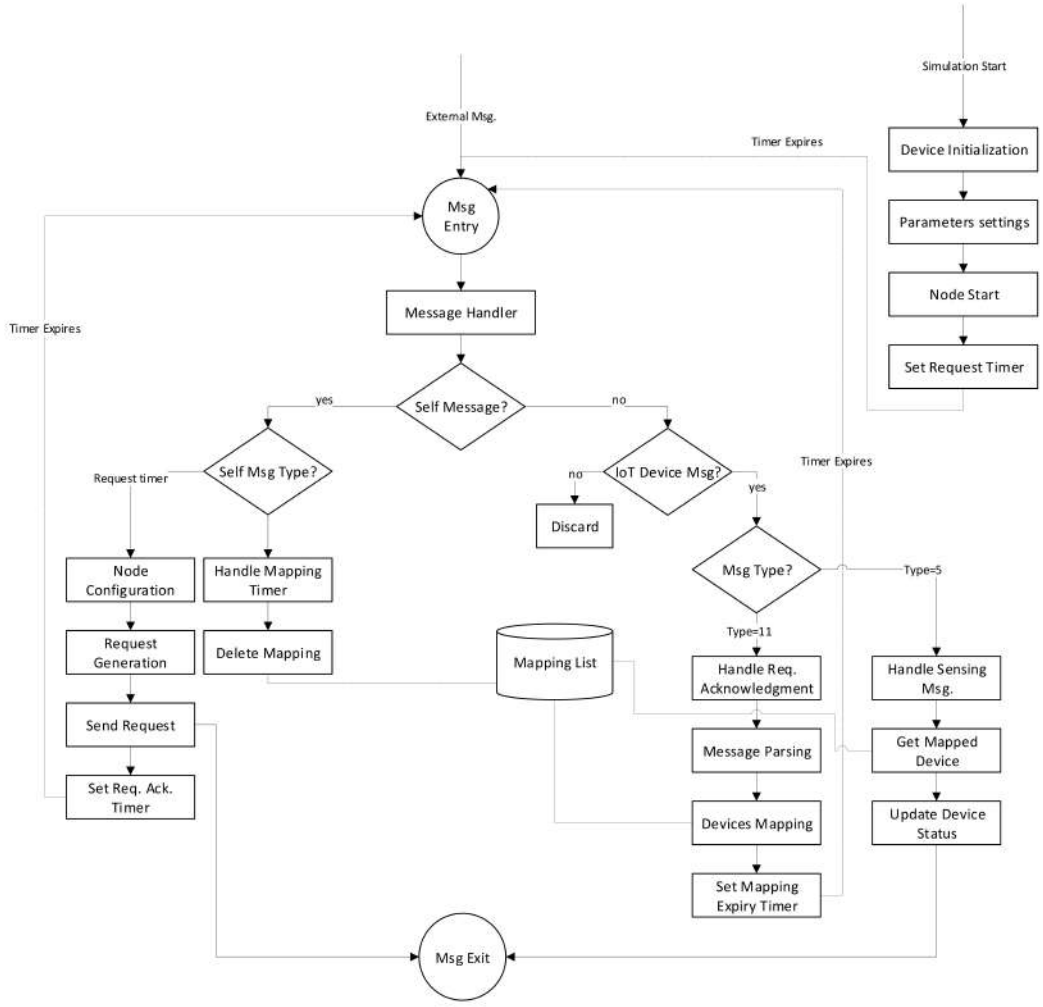


Figure 17. Working flow diagram of an IoT client device protocol for virtual IoT network simulation.

For the sake of illustration, a graphical view of nodes internal layered architecture is highlighted in Figure 18 (red rectangle for virtualization server, blue rectangle for client device, and yellow rectangle for IoT node). At the virtualization server, an instance of the *VirtualizationServerApp* protocol is used at the application layer. Likewise, instances of *IoTClientApp* and *IoTDeviceApp* protocol are used at the application layer of client device and IoT nodes, respectively.

In the beginning, IoT devices are associated with designated gateway nodes. Then, all IoT devices send registration requests to virtualization servers through the gateway node. IoT node color indicates its registration status i.e.,

- Red: request not yet sent,
- Yellow: request sent,
- Green: registered.

Registration request contains corresponding IoT device profile information. Upon registration, the virtualization server creates a virtual object (VO) for each IoT device to hold its profile information which is also used for onwards communication and control with/of corresponding IoT devices. Screen-shot given in Figure 18 is taken when the registration process of all IoT devices was completed and the activation command message is sent by a virtualization server to an IoT device. Furthermore, the tag on the application layer also indicates that 20 IoT devices are registered at the virtualization server (Registered VOs = 20). Similarly, the tag on the application layer of the IoT device is changed to Registered = yes indicating its registration with the server is successfully completed. After

registration, virtualization servers can receive requests from client devices and send various commands to registered IoT devices using its VO e.g., to get its operational status, to initiate data transmission, etc. Data transmission is initiated through a request sent by Client to *VirtualizationServerApp* using indicated part in an omnetpp.ini file as given in Figure 19.

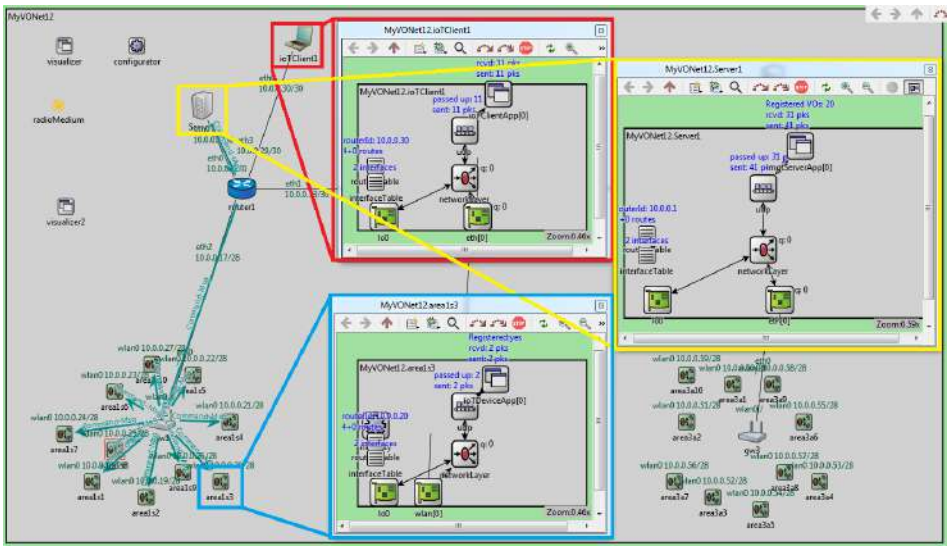


Figure 18. Client, IoT device and virtualization server internal protocol stack.

omnetpp.ini

```
##########################################################################################################
[Config VirtualIoTNet60-part5]
description = Testing Virtual IoT Network with 60 IoT devices, 01 Client and 01 Virtualization Server using One 2 One Scenarios
network = My_Virtual_IoT_Net

#Virtualization Server Configuration
*.Server1.numVirtualServerApps = 1
*.Server1.virtualServerApp[0].typename = "VirtualizationServerApp"
*.Server1.virtualServerApp[0].localPort=1000

*.Server1.virtualServerApp[0].destPort=0
*.Server1.virtualServerApp[0].messageLength=0B
*.Server1.virtualServerApp[0].sendInterval=0
*.Server1.virtualServerApp[0].dataRate=0
*.Server1.virtualServerApp[0].numActiveDevices=0

#IoT Client Device Configuration
*.ioTClient1.numClientApps = 1
*.ioTClient1.ioTClientApp[0].typename = "IoTClientApp"
*.ioTClient1.ioTClientApp[0].VirtualizationServerAddress="Server1"
*.ioTClient1.ioTClientApp[0].localPort=1000

*.ioTClient1.ioTClientApp[0].reqStartTime=2s
*.ioTClient1.ioTClientApp[0].reqStopTime=20s
*.ioTClient1.ioTClientApp[0].dataSendTo="One2One"
*.ioTClient1.ioTClientApp[0].reqDataRate=100
*.ioTClient1.ioTClientApp[0].reqNumActiveDevices=5
*.ioTClient1.ioTClientApp[0].reqMapping="VS01>VA01,VS02>VA02,VS03>VA03,VS04>VA04,VS05>VA05"

*.ioTClient1.ioTClientApp[0].destPort=0
*.ioTClient1.ioTClientApp[0].messageLength=0B
*.ioTClient1.ioTClientApp[0].sendInterval=0

#IoT Sensors Configuration
*.area1s*.numIoTApps =  1
*.area1s*.ioTDeviceApp[0].typename = "IoTDeviceApp"
*.area1s*.ioTDeviceApp[0].VirtualizationServerAddress="Server1"
*.area1s*.ioTDeviceApp[0].LocationName="Area1"
*.area1s*.ioTDeviceApp[0].localPort=1000
*.area1s*.ioTDeviceApp[0].DeviceClass="Sensor"
*.area1s*.ioTDeviceApp[0].DeviceType="Temperature"
*.area1s*.ioTDeviceApp[0].destPort=0
*.area1s*.ioTDeviceApp[0].messageLength=0B
*.area1s*.ioTDeviceApp[0].sendInterval=0

#IoT Actuators Configuration
*.area2a*.numIoTApps  =  1
*.area2a*.ioTDeviceApp[0].typename = "IoTDeviceApp"
*.area2a*.ioTDeviceApp[0].VirtualizationServerAddress = "Server1"
*.area2a*.ioTDeviceApp[0].LocationName="Area2"
*.area2a*.ioTDeviceApp[0].localPort=1000
*.area2a*.ioTDeviceApp[0].DeviceClass="Actuator"
*.area2a*.ioTDeviceApp[0].DeviceType="Fan"
*.area2a*.ioTDeviceApp[0].destPort=0
*.area2a*.ioTDeviceApp[0].messageLength=0B
*.area2a*.ioTDeviceApp[0].sendInterval=0
```

Figure 19. Simulation configuration for virtualization server, IoT client, sensors and actuators.

5. Simulation Design and Results

In the following sub-sections, we present our simulation settings and evaluation results of the proposed IoT network virtualization scheme. Among the various use case scenarios, we have selected a simplified one-to-one scenario where a virtualized IoT network is formed to establish a dynamic one-to-one communication link between selected sensor and actuators. A detailed discussion about selected scenarios' configuration and network parameters are presented in the next sub-section.

5.1. Simulation Scenario and Parameters

We performed a set of experiments with a fixed number of IoT devices in the network with varying packet sending rates. Figure 20 shows the simulation scenario used in these experiments. We have considered two domains (A and B) with 30 IoT devices in each domain. All IoT devices in a particular domain are connected with the virtualization server through a local gateway node using a wireless link with data rate 54 Mbps. Gateway nodes are connected to the virtualization server over a 100 Mbps wired link through intermediate routers over the internet.

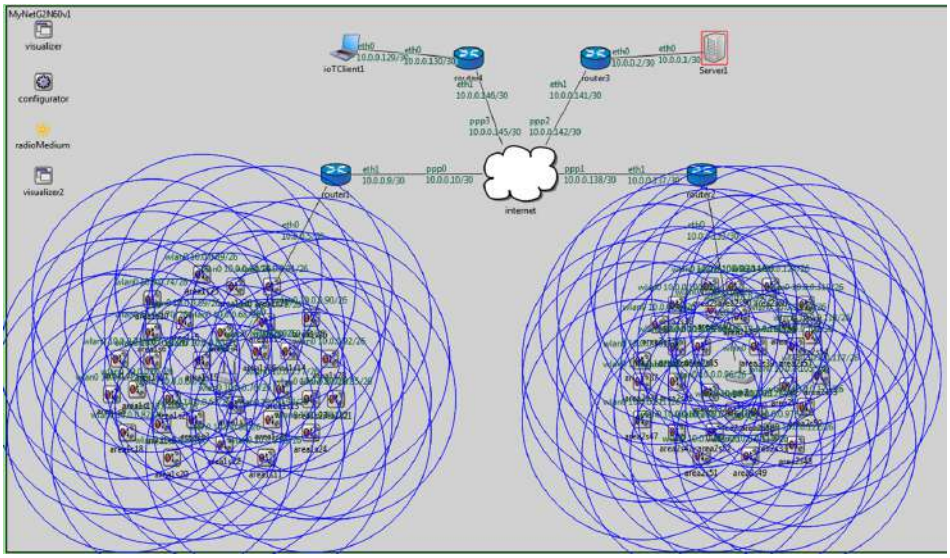


Figure 20. An IoT network virtualization simulation scenario in OMNET++.

During simulation start-up, all IoT devices send a registration request containing its profile information to a pre-configured virtualization server. Virtual objects are created for every connected IoT device. Afterwards, client node "ioTClient1" sends a request to the virtual server for the generation of a virtualized IoT network. The client desired topology is created by creating a virtual IoT network among selected virtual objects. In these experiments, a simple use case scenario is simulated by creating five dynamic connections (one-to-one) between five pairs of sensors and actuators. Sensors from domain A are connected to the actuators in domain B. Afterwards, sensors are activated to start sending data at a specified data rate that is received by corresponding virtual objects and then forwarded to associated actuators through device mapping. Five flows are generated to establish one-to-one communication between sensors from domain A to actuators in domain B. Results are collected with varying data sending rates of 80, 100 and 120 packets/s per flow. The packet size used in these experiments is 1000 Bytes. Table 4 presents the configuration for various parameters used in simulation.

We have selected three parameters for performance analysis of the proposed system through simulation i.e., end-to-end delay, throughput and packet delivery ratio to study the impact of varying data sending rate on these metrics. End-to-end delay is computed by measuring the total time taken for receiving data packets from the sensing device and sending actuation command to an associated actuating device. In other words, end-to-end delay is the difference between the time at which a sensing message was generated and the time at which an actuation command was received at a target

actuating device. Throughput is the total data received at target IoT devices in per unit time from all sources i.e., active IoT devices. Packet delivery ratio is the percentage ratio between the total number of received data packets and the total number of data packets generated in the network.

Table 4. Simulation parameters.

Parameter	Value/Range		
	IoT Device(s)	**Virtualization Server**	**Client Device**
Nodes count	60	1	1
Application-Layer	*IoTDeviceApp*	*VirtualizationServerApp*	*IoTClientApp*
Size of Packet	1000 Bytes	N/A	N/A
Sending rate of Packets (per node)	80, 100, 120 pkts/s	N/A	NA
Transport-Layer		UDP Protocol	
Routing-Layer		Static IP Protocol	
MAC-Layer	IEEE 802.11	Ethernet	Ethernet
Bit-Rate	54 Mbps	N/A	N/A
Wireless Communication Range	100 m	N/A	N/A
Size of Area		1000 m^2	
Nodes Mobility		Static	
Simulation Duration		20 s	

5.2. Results and Discussion

Impact of growing network size on throughput can be evaluated in two different ways: (a) increase data rate i.e., packets/sec while keeping the number of sources fixed; (b) increase number of sources i.e., active IoT device to generate more data in the network. We have used the first approach in this study to evaluate network performance in terms of throughput. For these experiments, we have considered a fixed network of 60 nodes with five sources i.e., active IoT devices. The same experiments were repeated (five times) with varying data rates of 80, 100 and 120 packets/sec and the average results for throughput and end-to-end delay are reported in Figures 21 and 22. These results show the observed variation in throughput and delay during simulation time. With five sources, data rate of 80 packets/sec and packet size of 1000 bytes, the total data generated in the network becomes $5 \times 80 \times 1000 \times 8 \div 10^6 = 3.2$ Mbps.

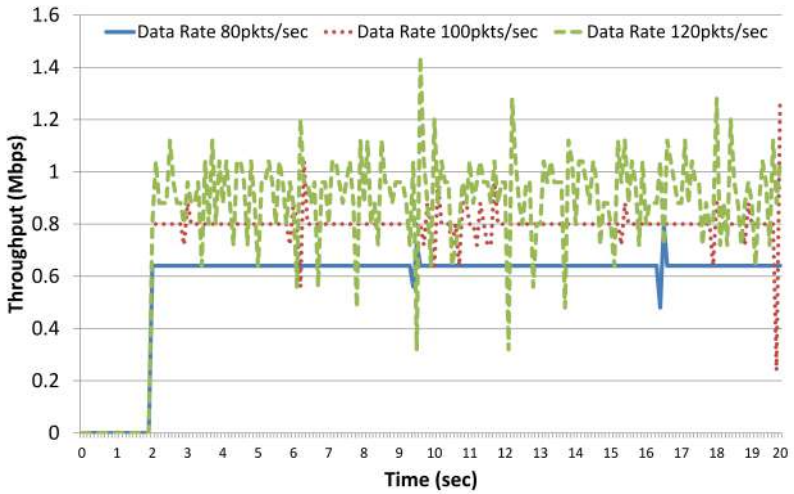


Figure 21. Throughput results.

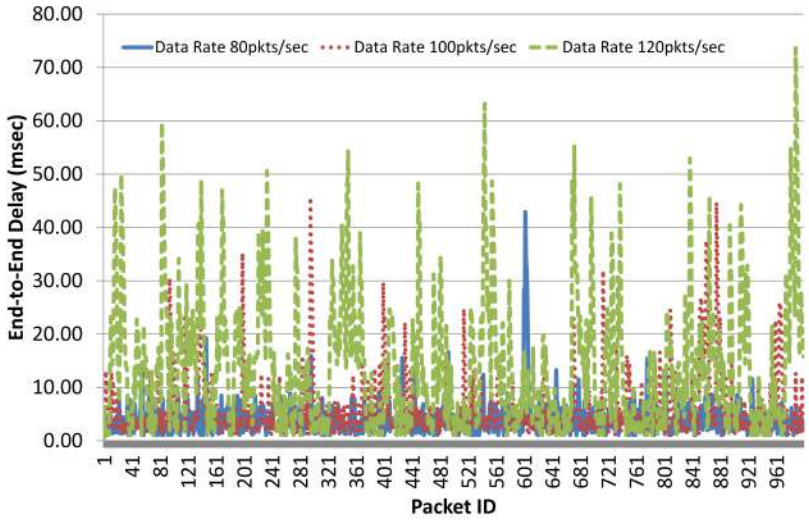


Figure 22. End-to-end delay results.

The mean value of throughput computed at all target actuating IoT devices through simulation was also around 3.2 Mbps with a standard deviation of 0.02 Mbps as shown in Table 5. It means that, with a data rate of 80 packets/sec, the packet delivery ratio in the network was 100%, which is confirmed from the packet delivery ratio graph given in Figure 23.

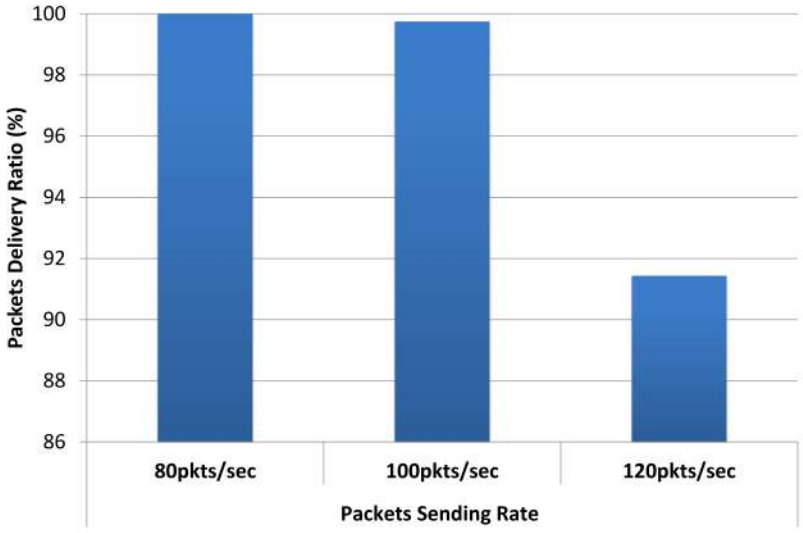


Figure 23. Packet delivery ratio results.

Moreover, throughput results with 80 packets/s are quite stable and there is no variation showing that all data packets are smoothly delivered. Minor variations in end-to-end delay results for data sending rate of 80 packets/s as shown in Figure 22 also indicate the there is no congestion in the network. The mean value of end-to-end delay for data sending rate of 80 packets/s was recorded as 3.94 ms with a standard deviation of 3.25 ms. When the packet sending rate was increased, i.e., 100 packets/s, we get data throughput around 3.99 Mbps, whereas data generated in the network was 4 Mbps. Variations in the throughput results for data sending rate of 100 packets/s given in Figure 21 also show that the network is getting congested and packet delivery ratio is slightly reduced as shown in Figure 23. Furthermore, end-to-end delay results for 100 packets/s given in Figure 22 also experience higher variations as compared to delay results for 80 packets/s. The mean value of end-to-end delay for data sending rate of 100 packets/sec was recorded as 6.38 ms with a standard deviation of 4.51 ms which is an almost 60% increase as compared to delay with data sending rate of

80 packets/s. Furthermore, an increase in data rate to 120 packets/sec results in mean throughput results of 4.39 Mbps, whereas data generated in the network is 4.8 Mbps. Higher fluctuations in throughput results are observed as shown in Figure 21. This is an indication of network congestion and Figure 23 shows significant degradation in packet delivery ratio with an increase in the data sending rate. Similarly, higher fluctuations were recorded in end-to-end delay results for 120 packets/s as given in Figure 22. The mean value of end-to-end delay for data sending rate of 120 packets/s was recorded as 13.46 ms with a standard deviation of 8.38 ms, which is an almost 240% and 110% increase as compared to delay with data sending rate of 80 and 100 packets/s. A statistical summary of throughput and end-to-end delay results are presented in Tables 5 and 6, respectively.

Table 5. Statistical results of throughput (Mbps).

	Packet Sending Rate		
Statistics	**80 pkts/s**	**100 pkts/s**	**120 pkts/s**
Mean	3.2	4	4.39
Stdev	0.02	0.07	0.21
Min.	0.48	0.24	0.32
Max.	0.8	1.28	1.44

Table 6. Statistical results of End-to-End Delay (msec).

	Packets Sending Rate		
Statistics	**80 pkts/s**	**100 pkts/s**	**120 pkts/s**
Mean	3.94	6.38	13.46
Stdev	3.25	4.51	8.38
Max.	42.9	45.67	73.65
Min.	0.13	1.05	0.13

Upon investigation, it was revealed from simulation trace files that, with an increase in the data sending rate, most of the packets get dropped at the MAC layer of IoT sensor nodes. Packets are dropped because nodes are unable to get access to the shared wireless channel due to contention. This is due to the fact that all IoT devices are using a single gateway node that is overloaded. Simulations results reveal that, with growing network size and data load, the gateway node becomes the performance bottleneck. To further validate this assertion, we have conducted another set of simulations with varying gateway nodes for similar network conditions. With an increase in packet sending rate, packet delivery ratio results are degraded for a single gateway node as shown in Figure 24. Significant improvement in packet delivery ratio can be observed when multiple gateway nodes are used for data rate 120 packets/s. Similarly, the mean value of end-to-end delay is reduced from 13.46 ms (with single gateway node) to 6.82 ms and 4.12 ms (with two and three gateway nodes), respectively as shown in Figure 25. In other words, a significant reduction in the end-to-end delay is observed with multiple gateway nodes i.e., 49.33% and 69.39% reduction with two and three gateway nodes, respectively, when compared to the results of the single gateway node. In extreme network conditions, we may have to sacrifice fewer bits, which is in fact worth nothing when compared to the flexibility and control offered by a proposed system over registered IoT devices.

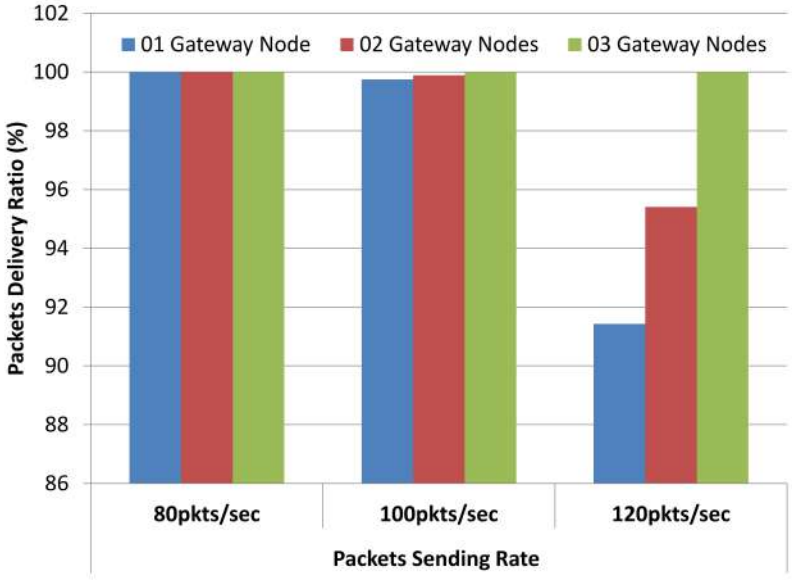


Figure 24. Packet delivery ratio results with varying gateway nodes.

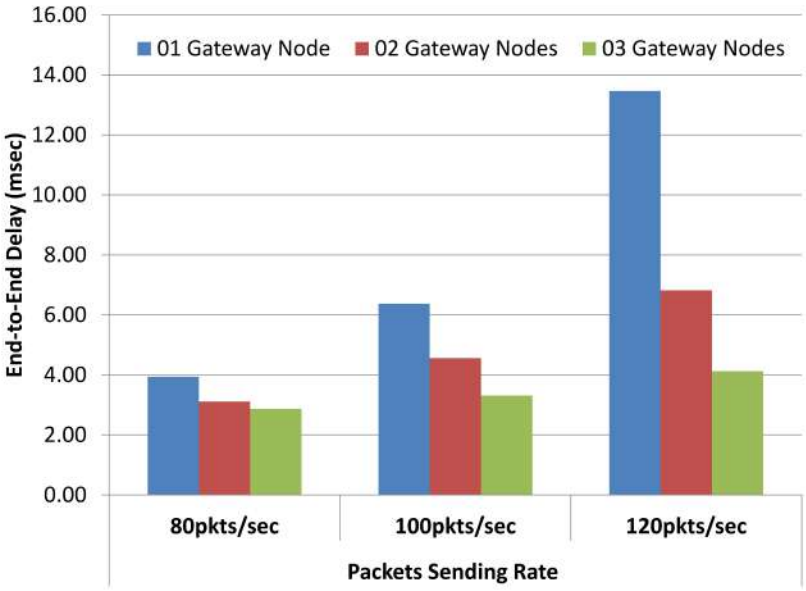


Figure 25. End-to-End delay results with varying gateway nodes.

5.2.1. Analysis of Virtual Objects' Resources Requirements

Typical representation of virtual object profile (IP Camera in this example) is given in Figure 9. The structure of virtual object profile will remain the same; however, memory requirement may vary depending upon the recorded properties of registered IoT devices. To study memory requirement at virtualization servers for registered virtual objects, we have conducted another set of experiments with growing network size. In these experiments, we consider seven different types of IoT devices i.e., serve motor, camera, temperature sensor, axis sensor, green led, buzzer, and light sensor where memory requirement for virtual object profile of each type device is 154, 170, 122, 122, 122, 122, 106 bytes, respectively. Results are collected from multiple simulation runs with a growing number of IoT devices from 200-to-1000 (step-size 200). In each experiment, IoT devices of aforementioned types were created through uniform distribution. After completion of registration process of all IoT devices at the virtualization server, we have recorded memory requirements for holding virtual objects' profile information and the average results are presented in Figure 26.

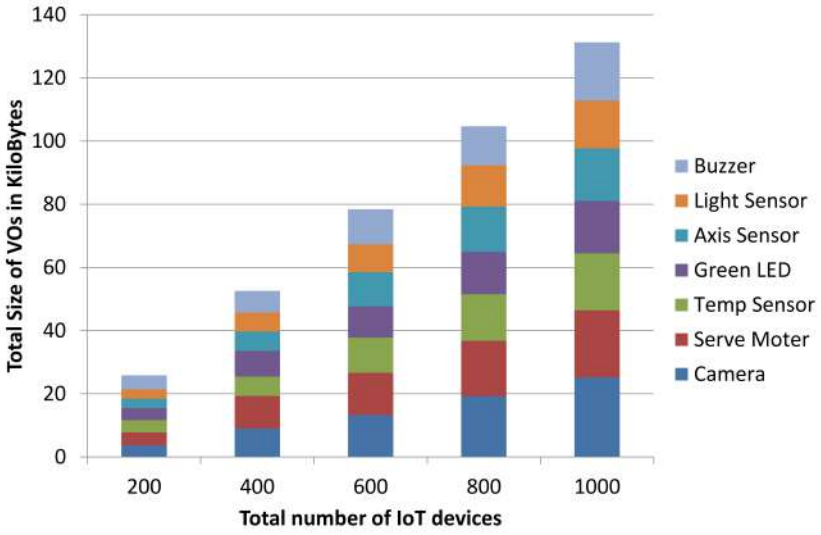


Figure 26. Analysis of memory requirements with a growing number of IoT devices.

It can observed that, with a growing number IoT devices in the network, memory requirements at a virtualization server also exhibit linear growth. However, depending upon the type of operation that we want to perform, computation requirements will be completely different. In these experiments, we have considered simple operations having computation complexity between $O(n)$ and $O(n^2)$. In the future, we will implement complex operations that will require more computation resources and conventional approaches will not be scalable when performed on a large scale with millions of registered IoT devices. This will certainly require intelligent and distributed solutions.

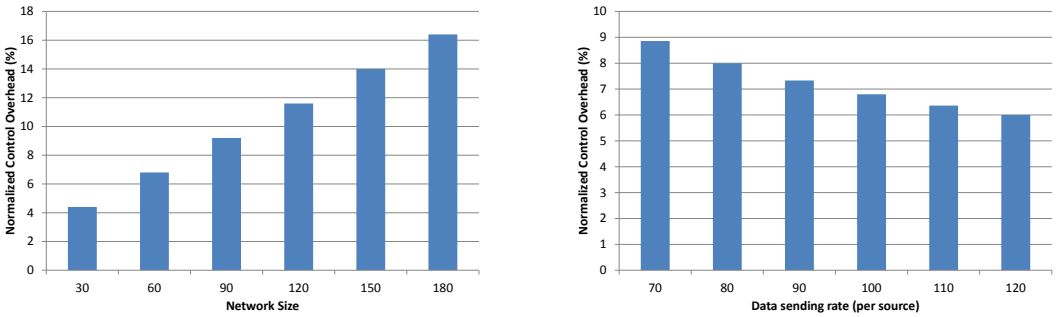
5.2.2. Analysis of Control Overhead

The proposed system generates various control packets in the network that includes initial registration packets, a command message to initiate and terminate various active flows in the network, and periodic status inquiry messages for registered IoT devices. To analyze overhead traffic due to a centralized virtualization server, we have conducted several sets of experiments. Experiments are performed with varying data sending rates and network size. We have used a normalized control overhead (NCO) metric to quantify network overhead of the proposed system. NCO is the ratio between total number of control packets and total number of data packets generated in the network. Mathematically, NCO can be computed as below:

$$NormalizedControlOverhead(\%) = \frac{Control packets}{Data packets} \times 100. \quad (1)$$

As given in Equation (1), this metric gives us the overhead in terms of control packets required for generating 100 data packets. Control packets generated in the network mainly depends upon the number of active flows and network size, while NCO also depends upon two factors i.e., data packets and control packets. NCO will increase if the network size is growing without an increase in the data sending rate or active flows. The same behavior can be observed in the experimental results presented in Figure 27a. The data sending rate and number of active source nodes are kept fixed; therefore, data packets generated in the whole network remain the same. However, the number of network control packets will increase with growing network size. Therefore, linear growth in NCO was observed with growing network size. With a network of size 180 IoT devices, the recorded value of NCO i.e., 16 (approximately) indicate that a proposed system generates around 16 control packets for every 100 data packets. However, with growing network size, an increase in data rate is also expected, which will then compensate for increasing control overhead. This assertion is validated through another set of experiments with a network of 60 IoT devices and a growing data sending rate. More data packets

are injected into the network with an increase in data sending rate and the number of control packets remaining the same. This results in a decrease in NCO as shown in Figure 27b.



(**a**) Normalized control overhead with growing network size. (**b**) Normalized control overhead with increasing data rate.

Figure 27. Simulation results with five active sources sending data@100 packets/s.

6. Conclusions and Future Work

This paper presents an idea of IoT network virtualization in the cloud environment. We have conducted performance analysis of the proposed system by implementing three application layer protocols in an OMNeT++ simulator. Various experiments are conducted with varying network sizes and packet sending rates. Simulation results indicate that, when the network traffic grows beyond a certain threshold, gateway nodes then become the performance bottleneck. Utilization of multiple gateway nodes can significantly improve the network performance in such conditions. Other solutions include improvement in wireless connectivity protocols for IoT devices. Traffic congestion can be avoided by dynamically adjusting the sensors' data sending rate through performing optimization at the virtualization server. In the proposed system, we have used simple virtual objects to create a virtual IoT network by performing a rule-based operation. In the future, we will extend this work by enhancing the capabilities of virtual objects for supporting complex interconnection among IoT devices through logical and service objects based on intelligent processing along with historical data stored in the data store. We are also looking forward to integrating this system with an IoT based application development tool and incorporating a security module. This will enable common users to rapidly build their customized IoT based applications by a quick selection of desired devices and services.

Author Contributions: I.U. implemented the protocols in OMNet++, performed simulations and results analysis, conducted the experiments and did a paper writeup. S.A. assisted with proposed system modeling and performance analysis. F.M. assisted in proposed system implementation and did a paper review and editing. D.H.K. conceived of the overall idea of IoT network virtualization in the cloud environment and supervised this work. All authors contributed to this paper.

Funding: This research was supported by Energy Cloud R&D Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT (2019M3F2A1073387).

Acknowledgments: This research was supported by Energy Cloud R&D Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT (2019M3F2A1073387), and this research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2019-2014-1-00743) supervised by the IITP(Institute for Information & communications Technology Planning & Evaluation). Any correspondence related to this paper should be addressed to Dohyeun Kim.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Miorandi, D.; Sicari, S.; De Pellegrini, F.; Chlamtac, I. Internet of things: Vision, applications and research challenges. *Ad Hoc Netw.* **2012**, *10*, 1497–1516. [CrossRef]

2. Tao, M.; Ota, K.; Dong, M.; Qian, Z. AccessAuth: Capacity-aware security access authentication in federated-IoT-enabled V2G networks. *J. Parallel Distrib. Comput.* **2018**, *118*, 107–117. [CrossRef]
3. Tao, M.; Ota, K.; Dong, M. Locating Compromised Data Sources in IoT-enabled Smart City: A Great-Alternative-Region-based Approach. *IEEE Trans. Ind. Inform.* **2018**, *14*, 2579–2587. [CrossRef]
4. Madria, S.; Kumar, V.; Dalvi, R. Sensor cloud: A cloud of virtual sensors. *IEEE Softw.* **2014**, *31*, 70–77. [CrossRef]
5. Ullah, I.; Sohail Khan, M.; Kim, D. IoT Services and Virtual Objects Management in Hyperconnected Things Network. *Mob. Inf. Syst.* **2018**, *2018*, 2516972. [CrossRef]
6. Bhattacharya, A.; Fernando, M.S.; Dasgupta, P. Community Sensor Grids: Virtualization for sharing across domains. In Proceedings of the First Workshop on Virtualization in Mobile Computing, Breckenridge, CO, USA, 17 June 2008; pp. 49–54.
7. Mazzei, D.; Montelisciani, G.; Fantoni, G.; Baldi, G. Internet of Things for designing smart objects. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Korea, 6–8 March 2014; pp. 293–297.
8. Khan, M.S.; Kim, D. DIY interface for enhanced service customization of remote IoT devices: A CoAP based prototype. *Int. J. Distrib. Sens. Netw.* **2015**, *11*, 542319. [CrossRef]
9. Kim, H.; Feamster, N. Improving network management with software defined networking. *IEEE Commun. Mag.* **2013**, *51*, 114–119. [CrossRef]
10. Ogrodowczyk, Ł.; Belter, B.; LeClerc, M. IoT ecosystem over programmable SDN infrastructure for smart city applications. In Proceedings of the 2016 Fifth European Workshop on Software-Defined Networks (EWSDN), The Hague, The Netherlands, 10–11 October 2016; pp. 49–51.
11. Tao, M.; Ota, K.; Dong, M. Ontology-based data semantic management and application in IoT-and cloud-enabled smart homes. *Future Gener. Comput. Syst.* **2017**, *76*, 528–539. [CrossRef]
12. Virtual Sensor Network. 2018. Available online: https://en.wikipedia.org/wiki/Virtualsensornetwork (accessed on 16 April 2018).
13. OverlayNetwork. 2018. Available online: http://en.wikipedia.org/wiki/Overlaynetwork (accessed on 16 April 2018).
14. Waharte, S.; Xiao, J.; Boutaba, R. Overlay wireless sensor networks for application-adaptive scheduling in WLAN. In *Proceedings of the IEEE International Conference on High Speed Networks and Multimedia Communications*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 676–684.
15. Sen, A.; Modekurthy, V.P.; Dalvi, R.; Madria, S. A sensor cloud test-bed for multi-model and multi-user sensor applications. In Proceedings of the 2016 IEEE Wireless Communications and Networking Conference (WCNC), Doha, Qatar, 3–6 April 2016; pp. 1–7.
16. Fazio, M.; Puliafito, A. Cloud4sens: A cloud-based architecture for sensor controlling and monitoring. *IEEE Commun. Mag.* **2015**, *53*, 41–47. [CrossRef]
17. Giezeman, W.; Stokking, J. The Things Network, 2016. Available online: https://www.thethingsnetwork.org/ (accessed on 14 June 2019).
18. Delgado, C.; Canales, M.; Ortín, J.; Gállego, J.R.; Redondi, A.; Bousnina, S.; Cesana, M. Energy-aware dynamic resource allocation in virtual sensor networks. In Proceedings of the 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2017; pp. 264–267.
19. Lemos, M.; Carvalho, C.; Rabelo, R.; Mendes, D.; Holanda Filho, R. An algorithm based on ant colony optimization for provisioning virtual sensor in sensor cloud. In Proceedings of the 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Banff, AB, Canada, 5–8 October 2017; pp. 2897–2902.
20. Lemos, M.; de Carvalho, C.; Lopes, D.; Rabelo, R.; Holanda Filho, R. Reducing Energy Consumption in Provisioning of Virtual Sensors by Similarity of Heterogenous Sensors. In Proceedings of the 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), Taipei, Taiwan, 27–29 March 2017; pp. 415–422.
21. Bousnina, S.; Cesana, M.; Ortín, J.; Delgado, C.; Gállego, J.R.; Canales, M. A greedy approach for resource allocation in Virtual Sensor Networks. In Proceedings of the 2017 IEEE Wireless Days, Porto, Portugal, 29–31 March 2017; pp. 15–20.
22. Faghih, M.M.; Moghaddam, M.E. SOMM: A new service oriented middleware for generic wireless multimedia sensor networks based on code mobility. *Sensors* **2011**, *11*, 10343–10371. [CrossRef] [PubMed]

23. Alam, S.; Chowdhury, M.M.; Noll, J. Virtualizing sensor for the enablement of semantic-aware internet of things ecosystem. *Int. J. Des. Anal. Tools . Integr. Circuits Syst.* **2011**, *2*, 41.
24. Harvan, M. A 6lowpan Implementation for TinyOS 2.0. *6. Fachgespräch Sensornetzwerke* **2007**, *802*, 109.
25. Raveendranathan, N.; Galzarano, S.; Loseu, V.; Gravina, R.; Giannantonio, R.; Sgroi, M.; Jafari, R.; Fortino, G. From modeling to implementation of virtual sensors in body sensor networks. *IEEE Sens. J.* **2012**, *12*, 583–593. [CrossRef]
26. Levis, P.; Culler, D. Maté: A tiny virtual machine for sensor networks. *ACM Sigplan Not.* **2002**, *37*, 85–95. [CrossRef]
27. Leontiadis, I.; Efstratiou, C.; Mascolo, C.; Crowcroft, J. SenShare: transforming sensor networks into multi-application sensing infrastructures. In *Proceedings of the European Conference on Wireless Sensor Networks*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 65–81.
28. Alam, S.; Chowdhury, M.M.; Noll, J. Senaas: An event-driven sensor virtualization approach for internet of things cloud. In Proceedings of the 2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications (NESEA), Suzhou, China, 25–26 November 2010; pp. 1–6.
29. Bose, S.; Gupta, A.; Adhikary, S.; Mukherjee, N. Towards a sensor-cloud infrastructure with sensor virtualization. In Proceedings of the Second Workshop on Mobile Sensing, Computing and Communication, Hangzhou, China, 22 June 2015; pp. 25–30.
30. Corcoran, P.M. Mapping home-network appliances to TCP/IP sockets using a three-tiered home gateway architecture. *IEEE Trans. Consum. Electron.* **1998**, *44*, 729–736. [CrossRef]
31. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
32. Salman, O.; Elhajj, I.; Kayssi, A.; Chehab, A. Edge computing enabling the Internet of Things. In Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 14–16 December 2015; pp. 603–608.
33. Khan, I.; Belqasmi, F.; Glitho, R.; Crespi, N.; Morrow, M.; Polakos, P. Wireless sensor network virtualization: A survey. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 553–576. [CrossRef]
34. Islam, M.M.; Hassan, M.M.; Lee, G.W.; Huh, E.N. A survey on virtualization of wireless sensor networks. *Sensors* **2012**, *12*, 2175–2207. [CrossRef] [PubMed]
35. Bizanis, N.; Kuipers, F.A. SDN and virtualization solutions for the Internet of Things: A survey. *IEEE Access* **2016**, *4*, 5591–5606. [CrossRef]
36. Flauzac, O.; Gonzalez, C.; Hachani, A.; Nolot, F. SDN based architecture for IoT and improvement of the security. In Proceedings of the 2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Gwangiu, Korea, 24–27 March 2015; pp. 688–693.
37. Omnes, N.; Bouillon, M.; Fromentoux, G.; Le Grand, O. A programmable and virtualized network & IT infrastructure for the internet of things: How can NFV & SDN help for facing the upcoming challenges. In Proceedings of the 2015 18th International Conference on Intelligence in Next, Generation Networks (ICIN), Paris, France, 17–19 February 2015; pp. 64–69.
38. Stankovic, J.A. Research directions for the internet of things. *IEEE Internet Things J.* **2014**, *1*, 3–9. [CrossRef]
39. Varga, A. OMNeT++. In *Modeling and Tools for Network Simulation*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 35–59.
40. Varga, A. INET Framework for the OMNeT++ Discrete Event Simulator, 2012. Available online: https://inet.omnetpp.org (accessed on 3 February 2018)

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).