Journal of Ambient Intelligence and Humanized Computing manuscript No.
(will be inserted by the editor)

LISA 2.0: Lightweight Internet of Things Service Bus Architecture Using Node Centric Networking

Behailu Negash[†*] · **Amir-Mohammad Rahmani**[†‡] · **Tomi Westerlund**[†] · **Pasi Liljeberg**[†] · **Hannu Tenhunen**[†‡]

Received: date / Accepted: date

Abstract Internet of Things (IoT) technologies are advancing rapidly and a wide range of physical networking alternatives, communication standards and platforms are introduced. However, due to differences in system requirements and resource constraints in devices, there exist variations in these technologies, standards, and platforms. Consequently, application silos are formed. In contrast to the freedom of choice attained by a range of options, the heterogeneity of the technologies is a critical interoperability challenge faced by IoT systems. Moreover, IoT is also limited to address new requirements that arise due to the nature of the majority of smart devices. These requirements, such as mobility and intermittent availability, are hardly satisfied by the current IoT technologies following the end-to-end model inherited from the Internet. This paper introduces a lightweight, distributed, and embedded service bus called LISA which follows a Node Centric Networking architecture. LISA is designed to provide interoperability for resource-constrained devices in IoT. It also enables a natural way of embracing the new IoT requirements, such as mobility and intermittent availability, through Node Centric Networking. LISA provides a simple application programming interface for developers, hiding the variations in platform, protocol or physical network, thus facilitating interoperability in IoT systems. LISA is inspired by Network on Terminal Architecture (NoTA), a service centric open architecture originated by Nokia Research Center. Our extensive experimental results show the efficiency and scalability of LISA in providing a lightweight interoperability for IoT systems.

Keywords LISA · IoT · Interoperability · NoTA · NCN · Mobility

†Department of Information Technology
University of Turku, Turku, Finland
E-mail: *behneg@utu.fi

‡Department of Industrial and Medical Electronics
School of ICT
KTH Royal Institute of Technology, Stockholm, Sweden

1 Introduction

The Internet has evolved through different changes that shaped it to the current vast hub of knowledge where more number of devices are connected than the human population (Evans, 2011). One of the changes is the shift in the requirement of computer networks from the need to share hardware resources (Roberts and Wessler, 1970) to information sharing. This demand for information sharing through a large network of computers across the globe sustained researches in academia and military that contributed to the birth of the Internet. It is estimated that more than 50 Billion devices will be connected by 2020 (CERP-IOT, 2010). A large proportion of these devices will be small embedded components which will be identified uniquely and are supposed to interact with other devices. This network of embedded devices is referred as Internet of Things (IoT) (Tan and Wang, 2010). It is believed that enormous economic and social benefits can be gained by utilizing IoT in different application domains.

To achieve the envisioned benefits of IoT, the smart devices have to be able to communicate and exchange information in an interoperable way. To this end, devices need to follow a consistent network interface implementation and use the same communication standards. This is achieved by using a common standard

for the Internet. However, due to limited memory, processing capacity and available power in the majority of IoT components, it is not feasible to utilize the same standards proposed for the Internet. Therefore, many new technologies, standards, and platforms are introduced to work in various operating conditions, and for different requirements, each creating silos of applications. To overcome the heterogeneity in IoT, few organizations and alliances have contributed to provide middleware for interoperability of the common protocols and platforms. However, these frameworks do not address the vast majority of the IoT devices which are tightly resource-constrained.

On the other hand, end-to-end arguments (Saltzer et al., 1984) introduced three decades ago direct the design of computer networks and subsequently the Internet architecture up to now. Based on the end-to-end arguments, the networking function is expected to locate and address the end hosts as a mandatory step for data transfer. Different system requirements were listed and the networking function was designed to meet these requirements accordingly. Therefore, application specific requirements are built on top of this core network function that is focused on data transfer. However, wide usage scenarios and various application contexts of computer networks, brought new requirements that led to different modifications of core data transfer function of network services. One of these modifications is the introduction of Network Address Translation (NAT) (Egevang and Francis, 1994) to overcome the shortage of IPv4 addresses. By introducing NAT, local addresses of communicating hosts are hidden behind a router which takes care of the external communication representing the local hosts, which effectively forms a hierarchy of addresses. Domain Name System (DNS) (Mockapetris, 1987) is a system which is used to translate Internet domain names in to machine network address. It is another modification to the initial Internet design to facilitate access for non-technical users and a means to easily remember meaningful names (than sequence of numbers) which identify an Internet service. Security is another aspect that triggered the motivation for a new set of requirements in the Internet architecture. The introduction of IoT adds a completely new dimension to the set of requirements thereby requiring a new mindset than the old end-to-end arguments.

In an effort to address the new network requirements, Blumenthal *et al.* (Blumenthal and Clark, 2001) introduce a new perspective about the design of the Internet. Information-Centric Networking is one of the most novel Internet architectures considering the Internet of Things as a typical application scenario. Content-Centric Networking (CCN) (Jacobson et al., 2009) is a special type of Information Centric Networking where contents are named and used as the center of communication than the host machines. Instead of finding the location of information, the information itself is searched and routed in CCN. Looking closely at the original requirements of computer networks (sharing hardware resources), it is logical to address the hardware devices for communication. A device interested to communicate with another node sends a request to the destination address. However, in the context of information sharing, a device is only interested in the specific requested information and its validity, not where it is located. Therefore, CCN overcomes the unnecessary routing of destination addresses by replacing it with content name as the main requested entity.

In parallel to the changes in the network architecture, the applications that run on a network service have also faced major changes. Prior to the introduction of distributed information systems, simpler applications designed for specific tasks have been developed independently. As the requirements of these systems increase, larger, distributed and integrated systems were demanded. One solution to integrate such independent applications is to enable the smaller applications to expose their functionality as a service to be consumed by others, known as Service Oriented Architecture (SOA) (Sarang, 2007). However, the implementation of SOA leads to a large amount of interaction between these services. In addition, each service instance has to be aware of the location of the other end as well as interfaces exposed during communication. Enterprise service bus (ESB) (Keen et al., 2004) gives an infrastructure for SOA implementation for managing the communication between services. It also provides a common medium for communication between different standards to facilitate interoperability.

This paper introduces a lightweight IoT service bus architecture using node centric networking, called LISA 2.0, for interoperability of a wide range of IoT devices. It is a major extension of our preliminary implementation of the LISA presented in (Negash et al., 2015). It provides a uniform, socket like, application programming interface hiding the underlying physical network interface, protocol and platform differences. LISA is small enough to fit inside the network stack of small embedded operating systems for constrained devices and also scales for devices with less resource constraints. It takes advantage of an intermediate computing layer between the cloud and end devices to facilitate interoperability, mobility and allow low-power operation of devices. In addition, this paper introduces a new type of information centric networking implemented in LISA, known as Node Centric Networking (NCN). A node

is defined in this context as either a service-end handling requests or an application utilizing services in a SOA implementation, which is independent of the hardware device. Based on such definition of a node, two or more nodes can co-exist on a single device depending on the available resources (memory, processing capacity and power source). LISA is inspired by the Network on Terminal Architecture (NoTA), a service centric open architecture originated by Nokia Research Center. In short, the key features of LISA 2.0 are as follows:

- Being lightweight and targeting extremely resource constrained nodes with a few kilobytes of memory.
- Supporting low-power operations with an ultimate goal of supporting most common interconnection protocols for interoperability.
- Introducing a new node centric networking concept and a hierarchical addressing mechanism.
- Utilizing NCN addresses for inter-protocol message routing for interoperability.
- Leveraging an intermediate computing layer to support tiny nodes in overcoming resource constraints issues.

The following sections present the motivation for LISA and NCN (Section 2), the basis for the LISA design (Section 3) and the LISA Implementation (Section 4). Demonstration and evaluation of the middleware are presented in Section 5 while comparison with related work (Section 6) and conclusion (Section 7) are discussed at the end.

2 Motivation

One of the many application domains, where IoT is expected to considerably enhance user experience, is health and well-being. A specific application area of IoT in this regard is Ambient Assisted Living (AAL). AAL is a combination of different components where elderly or in-home patients are remotely monitored to be provided with a safe and convenient environment. Fig. 1 shows a simplified view of AAL, with one body temperature sensor, two thermostats in different rooms, and a central gateway for a smart home system. The AAL system in this scenario is an integration of two separate systems: a smart home system which is composed of smart lighting system, smart home appliances, smart security system being all connected to a central gateway, and a mobile health system which is built from a range of vital signal sensors connected to a smart phone. Consider a situation where the body temperature sensor sends a control signal to a thermostat to adjust the room temperature.

From the system described above, based on available resources, we can categorize the devices into three classes. The first class contains those devices, such as PC, smart phones and gateways, having multiple high speed processors with gigabytes of memory and multiple networking options. Those devices which have few megabytes of memory and used in different Industrial or home automation belong to the second class (such as thermostats, motion detectors, and smart switches). The last class contains devices which are too limited in processing power, memory and usually have a single network interface. These devices might operate on batteries which are meant to be used for a long period of time. Typical devices in this class include sensors (such as body temperature sensor) which could have resources as small as 8-bit micro controllers having 32 Kbyte of RAM and 512 Kbyte of flash memory.

To integrate these two systems and build the AAL scenario, first the body temperature sensor needs to have the same network interface as the thermostat. Second, the two devices have to use the same protocol. Third, the thermostat has to be in listening mode to receive the message and understand the content of the message unambiguously. In addition, when a patient moves from one room to another, the body temperature sensor needs to automatically roam the communication to the thermostat resided in the room the patient just moved in. Fulfilling these requirements is not a straightforward programming challenge that could be addressed by a single existing middleware. There are few choices of frameworks (Razzaque et al., 2016),(Derhamy et al., 2015) that could be used to solve a portion of the above requirements. However, these frameworks fail to address the resource constraints of the third class of devices and limit the mobility of the patient. The above example is a typical case of an IoT vision (Singh et al., 2014) limited by a new set of requirements that need to be addressed.

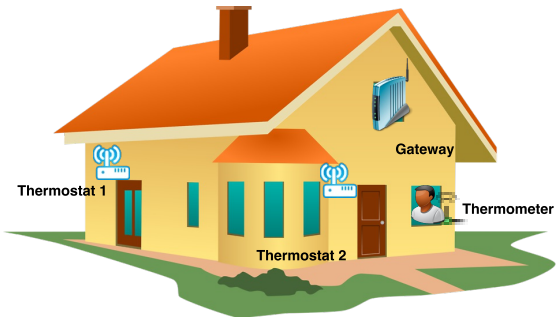

Fig. 1: Simplified AAL scenario

The main motivation of LISA comes from the IoT concerns that arise when mash-up of separate IoT systems is developed. Most of these requirements also exist in smaller IoT systems. The central element is interoperability. In IoT systems, interoperability is discussed in three layers; technical, syntactic, and semantic interoperability (IERC AC4, 2013). LISA is designed to address all the three interoperability layers. The other motivation is the mobility requirement of IoT systems, which is addressed by making the application and service-ends as central component of the communication in the NCN implementation in LISA. NCN is proposed based on modifications to the Internet architecture, such as CCN and TRIAD (A Scalable Deployable NAT-based Internet Architecture) (Cheriton and Gritter, 2000), that rely on name based routing. In addition, the majority of IoT components are constrained in terms of available memory, processing speed and power source. They are usually battery powered and need to efficiently manage the battery for a long period of time. Therefore, when devices are in idle state, they are supposed to go to sleep state. LISA is designed with the motivation of working within the limitation of these resource-constrained devices.

The target of LISA is to fill the interoperability gap by providing a simple API for programmers of such resource-constrained devices enabling developers to build a distributed and interoperable system. In the following section, we discuss the roots of overall architecture of LISA.

3 Foundations of LISA

Interoperability requirements are common in information systems. Some of the causes for differences in computing or communication are lack of standardization, differences in system requirement and application domain. Since the early days of distributed computing, there has been solutions proposed to overcome heterogeneity. LISA is a middleware that relies on basic principles of existing integration patterns and practices described in (Hohpe and Woolf, 2003). The basic technologies and patterns followed by LISA are discussed in the following sections.

3.1 Service Oriented Architecture and Enterprise Service Bus

It was briefly mentioned that LISA enables the implementation of Service Oriented Architecture (SOA) in IoT systems. SOA is an architectural style where individual functionality of a system are exposed as autonomous services so that consumers of a service can easily access it (Keen et al., 2004). This has been extensively used to solve the problem of interoperability in enterprise systems regardless of the platform on which the service runs. One example of such systems is shown in (Amicis et al., 2011), where SOA is used for interoperability of transport infrastructures. The services have defined interfaces which are described using standard specification such as web service description language (WSDL). The format of exchanged message is usually either XML or JSON (Keen et al., 2004). Moreover, Universal Description, Discovery and Integration (UDDI) is used to register web services so that client applications can easily look up the details of the service. Similar implementation exists in LISA, where services register to a central register for application nodes to discover their location and learn other important behaviours (such as the protocol it uses).

An Enterprise Service Bus (ESB) is, in the simplest form, a middleware that provides essential services to implement Service Oriented Architecture (SOA) (Keen et al., 2004). ESB enables different services, which use various incompatible platforms and languages, to communicate with each other. ESB provides services for secure communication channel and a way of translation between distinct protocols. ESB facilitates sharing of the interface exposed by a service in a SOA architecture to the clients interested in the service. In some configuration, ESB can be distributed (each host has a section of the ESB locally) or centralized. There are various open source and proprietary ESBs for the Enterprise domain (Alghamdi et al., 2010). However, most of these ESBs use XML as an exchange format and web services as end nodes. Unlike the enterprise ESBs, that run at higher abstraction level in the network stack and are heavy for resource constrained devices, LISA works at lower abstraction level and is very lightweight. In addition, LISA is a distributed service bus, enabling direct communication of a client and server over the bus.

3.2 Network on Terminal Architecture

Network on Terminal Architecture (NoTA) is a service based open architecture designed by Nokia Research Center (Binnema, 2009). The initial objective for the design of this architecture was to facilitate the development of mobile devices by minimizing the time required to integrate different modules. It also minimizes the coupling of system modules. The architecture has its roots in the area of Network on Chip (NoC) and web services as it focuses on interconnection of services and applications utilizing these techniques. It is a platform and protocol independent architecture. The refer-

ence implementation of NoTA, which was released as an open source implementation of this architecture, supports variety of network protocols and few platforms.

There are two type of nodes in NoTA; service nodes (SN) and application nodes (AN). The application node is a client that requests for a service from service node. The interaction of these nodes pass through a stack called the Device Interconnect protocol (DIP), shown in Fig. 2 (a). The DIP has two layers; High (H_IN) and Low (L_IN) interconnect layer (Binnema, 2009). Application and service nodes communicate either using message passing (in control plane) or a node streams data to a target (in data plane).

The High interconnect layer (H_IN), shown in Fig. 2 (b), is the device interconnect layer that is closer to the user application and it provides Berkeley socket (BSD) like interface for the programmers. H_IN is where the NoTA protocol is implemented with functionalities such as service discovery, activation and registration. It has two operating modes; a single process mode and a multi-process mode which is enabled running a NoTA daemon service. On the other end of H_IN, it communicates with the underlying layer, L_IN to interact with the specific network protocol.

The lower interconnect layer (L_IN) has two sub layers, L_INup and L_INdown, as shown in Fig. 2 (b). The upper layer (L_INup) communicates with H_IN and the lower layer (L_INdown) provides specific implementation of different protocols. Different L_INdown implementations are enabled for a specific node based on the available network interface and in some cases multiple L_INdown can be enabled at the same time.

NoTA also provides a special type of service node, which can be enabled depending on the size of the network. This service node is called Resource Manager (RM), which is used to handle dynamic discovery of services by application nodes. NoTA also provides a stub generator, which is an easy way of handling the process of creating and using NoTA sockets. The services in NoTA are described using an XML based service description file, similar to web service description language (WSDL). The format of messages in NoTA is different from that of web services. NoTA defines service signal types that specify message parameters for nodes to understand the content. Unfortunately, NoTA was stopped after six years of development and two updates of the reference implementation of the architecture. This work was started by exploring the option of using NoTA for IoT. Even though it was not possible to use NoTA for the targeted resource constrained implementation, NoTA has become an inspiration for this work. The service description technique and message format of NoTA are used in LISA as-is.

3.3 Content Centric Networking

To share hardware resources, every one of the participating devices needs an address to locate the other party; for instance, IP addresses are used to identify devices connected to the Internet. Two communicating devices use this IP address to send and receive messages. In the current Internet, however, it is mostly information or content that is shared rather than hardware resources. An alternative idea is proposed to embrace the changes to what is shared over the Internet. Content centric networking (CCN) (Jacobson et al., 2009) is one type of information centric networking. It is a novel network architecture modified from the current Internet architecture. CCN is aimed at minimizing the problems that arise due to the tight coupling of the current device addressing scheme with the initial purpose of a computer network. Palo Alto Research Center (PARC), the organization behind CCN, has an open source implementation of the protocol called Project CCNx for few supported platforms and application plug-in for media transfer.

In CCN architecture, contents are given unique names where consumers search for the content using this name. One of the widely used naming scheme in CCN is hierarchical naming. Parts of names indicate chain of relation in the network similar to path names in a file system. For CCN to work, it requires faster and more efficient routing algorithms. This is due to the fact that high volume of content is routed in a network containing few communicating entities. In addition, one of the targets of CCN is to secure the information that is communicated over the network instead of the network infrastructure itself. Therefore, a device interested in a certain content sends an interest request that could be handled by any other device containing that specific content. A new type of information centric networking is implemented in LISA.

3.4 Software Defined Networking

The focus of the network service has been to create a channel for data transfer without any dependence on the type of actual data exchanged. For the network service to achieve this, there are different configurations that has to be done by the network administrators. Individual network devices working at different layers of the protocol stack are configured in different ways depending on the device manufacturer. This creates a bottleneck for a network service to be agile and meet new application requirements. In addition, the data and control plan in the traditional network are merged in one device. The latest approach to

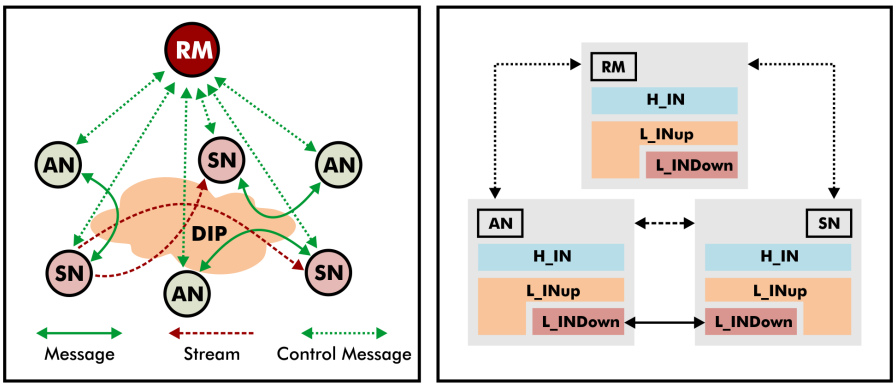

Fig. 2: (a). NoTA nodes and message types; (b) Inside NoTA DIP

overcome this problem, that enables a more agile and scalable network is called Software Defined Networking (SDN)(Kim and Feamster, 2013). In SDN, the network devices between two communicating devices are divided into control plane devices (called controllers) and data plane devices (called forwarding devices or switches). The controller is logically organized in a central manner whereas the switches are hierarchically organized and managed by controller. The controller has a high level overview of the network architecture and can provide a better decision on packets arriving at switches. Switches forward the new packets whose information does not exist in the local flow table to get instruction from controller. There are still a wide range of ongoing researches(Kreutz et al., 2015) in SDN in general. A framework proposed by (Jararweh et al., 2015) integrates the idea of SDN to solve the challenges of traditional architecture. In comparison to our NCN proposal in this paper, SDN is viewed as an enabler underlying infrastructure. There are some works (Nguyen et al., 2013) that research on the possibility of running information centric networking on top of SDN.

4 LISA: Lightweight IoT service bus

In this section, we explore the design details of LISA. As discussed in the previous sections, LISA is designed specifically for resource constrained devices. In most implementations, these devices use a lightweight operating system to take care of low level details so that user applications can focus on specific system requirements. There are many flavors of such operating systems; for instance Contiki, TinyOS, RIOT and FreeRTOS are widely used. Regardless of the internal design of these operating systems, they are all designed to be lightweight. One of the requirements of interoperability is the ability to use a combination of these operating systems in a large system. In addition, each device might also have a different network interface or utilize different protocols. LISA unifies these differences in platform and protocol levels by enabling the implementation of SOA. LISA is designed to be portable between these operating systems with minimal configuration. In some cases however, there are applications that run without any operating system. One goal is to enable a bare metal implementation in future releases. The initial version of LISA, however, targets RIOT, discussed in the following section.

4.1 Target platform

RIOT is a micro kernel based, real-time, multi-threaded and modular operating system specifically designed to take resource-constrained nodes into consideration. It is designed to bridge the gap between the full-fledged operating systems (e.g. Linux and Windows) which are easier to program and the smaller operating systems for sensor nodes (e.g. Contiki and TinyOs) by providing easy programmability for the lower end. It supports standard C programming and provides inter-process communication facilities with partial POSIX compliance. It has a lightweight network stack with 6LoWPAN and Routing Protocol for Low-Power and Lossy Networks (RPL) support. The overall features provided for module development and easy portability to multiple boards and CPUs attracted us to make RIOT the first target platform to test the concept of LISA. As shown in Fig. 3, LISA runs on top of the transport layer and can be considered as an alternate socket layer which handles user application requests for network services.

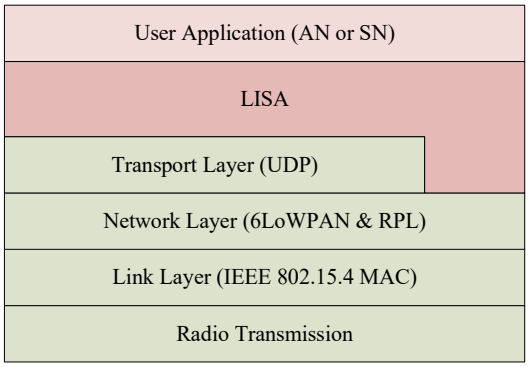

Fig. 3: LISA in the network stack of RIOT

4.2 High level architecture

To achieve interoperability among protocol and platform silos, different approaches has been taken. The first approach is to have a single standard to overrule all the others. This approach so far has been tried by many industry alliances for IoT, some of which are specific to application areas (such as home automation). The second approach is to provide a means of translation of the transmitted information between different standards. So far, the first approach has resulted mostly in an additional competing standards instead of solving the interoperability problem. Our project approaches the problem of interoperability using the second method by providing a modular framework that scales from tiny devices, which operate within resource constraints, up to the high capacity processing devices.

Figure 4 shows the components of LISA at a high abstraction level. A user application on the top most part of the stack communicates with the core of LISA through a common programming interface (shown in figure as Application Interface). The application interface is intentionally made to resemble BSD sockets. The socket interface is a low level abstraction, to eliminate unnecessary high level abstractions that would make our middleware heavy. It is also familiar among programmers. More over, due to the fact that LISA was initially designed to work with NoTA, which has a similar interface, it helps programmers to have similar experience in both cases. The calls from the user application to the internal LISA are handled in the LISA core including the start-up procedure to setup the federated architecture. LISA Core is the heart of the framework which implements LISA protocol and other core services for connecting to a manager node. The core module utilizes the network services provided by the network stack and other important low level features built in the platform through the two interfaces, shown on the side and bottom of the stack, that enable swapping between different protocols and platforms. Different modules which are specific to the protocol or platform handle requests coming through the common interface. As a result, user applications can interoperate with others regardless of the underlying protocol used.

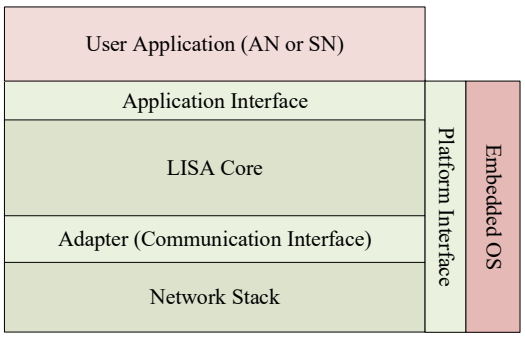

Fig. 4: High level components of LISA

4.3 Federated Nodes

LISA has three node types; application nodes (AN), service nodes (SN) and a manager node. LISA nodes are organized in a federated manner as shown in Fig. 5. That is, autonomous subnetworks of ANs and SNs are managed with a single manager node and a group of subnetworks form a bigger network. The bigger network is managed by a single main manager node elected from the subnetwork managers with the support of a cloud service. Since all manager nodes are connected to the cloud, communication between manager nodes can take through the cloud. Optionally, near by managers can communicate directly without the need to pass through the cloud. Following the node types, there are different levels of uptime. Application node is in a sleep state most of the time and initiates communication whenever it wakes up. Service nodes can also go to the sleep state, but they are active for longer time than application nodes. This effectively maps the three classes of devices discussed in previous section into the three node types. Application and service nodes receive manager advertisement which contains the address of the manager. Service nodes are identified with a unique name which is descriptive of the service it provides. Services register with the manager node as they wake up and a unique domain specific address is generated for the service. Application nodes request the manager for the address of a service during discovery phase. Once the address is resolved the application and service node can

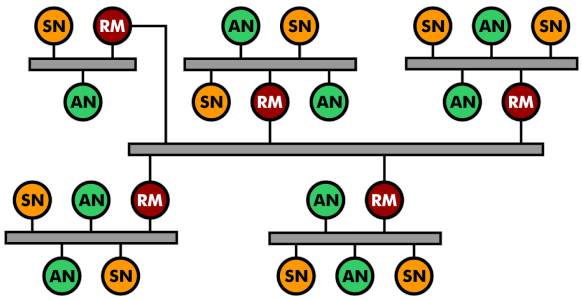

Fig. 5: Federated architecture of LISA

start to communicate directly, provided that they use the same protocol.

When there is a request for a service which cannot be found in the node register of the local manager (outside the subnetwork), the manager node represents the node in passing the request to the manager of the respective destination subnetwork. Service requests inside a subnetwork occur simply in a client-server fashion once the channel is established with the help of the manager. For ease of discussion, Fig. 5 shows five subnetworks each having a single manager (red nodes in Fig. 5) and communicating using a different protocol (such as 6LoWPAN or Bluetooth low Energy). This local manager is known as Home manager for the nodes in the subnetwork. Inside a given subnetwork, an application node can simply communicate with the service in a peer-to-peer fashion. However, if a node in one subnetwork, for instance using 6LoWPAN, requests a service which is using Bluetooth low energy, the managers of these sub networks act on behalf of the individual nodes in transferring messages.

4.4 Fog computing and LISA

The federated architecture discussed in the above section followed the function of the nodes, thereby enabling the classification of various devices into the three node types. Looking at another functional classification, LISA operates in two computing layers. Application and service nodes are in the bottom end device layer and the manager nodes are located at the edge of the network between the end devices and the cloud. To accommodate IoT requirements for interoperability, mobility and limited availability of devices, LISA uses the intermediate computing layer, known as Fog computing layer. Fog computing is a new computing paradigm extending the cloud computing concept, where the characteristics of the cloud computing are brought closer to the network edge where data is generated. Fog computing provides fast and real-time response to changes in reading values by end nodes. For instance, in the example discussed in the motivation section, the smart phone and smart home gateway constitute intermediate computing layer between the sensor devices and the cloud.

4.5 LISA protocol

Figure 6 shows the different protocols and the transition of the state of LISA nodes. At a higher level of classification, LISA has two types of messages; setup messages and user messages. Setup messages are used during the initial phase (such as discovery, registration and access request) of locating and authenticating nodes whereas user messages are application specific contents. There are five types of setup messages which are used as a standard for the bus to create the channel to another segment of the bus. These messages are used for service discovery, registration, application authentication, handshake of nodes and service deactivation. Each of these messages affect the state of the bus segment in the local node. Table 1 summarizes the messages and states in the LISA protocol.

A service node sends registration request message (SRP_REQ) to a manager in two conditions. First when it cannot find a manager address in its internal address log. The second condition is when the locally stored manager address is different from the one it currently receives through advertisement. If the registration is successful, the manager replies with registration confirmation (SRP_CNF). To allow mobility of devices, nodes are not configured with static addresses. Therefore, clients nodes have to discover for the current location and address of the service of interest. That is, the client node (either an application or service node) sends a discovery request (SDP_REQ) for the manager node using the service name. It is possible for two or more services to have the same service name as long as they provide functionally similar service. However the full ontology name constructed as an identifier in RDF format is unique for every node. Looking back at the motivational example, the two thermostats can have the same service name. If the manager locates the requested service, it replies a confirmation message (SDP_CNF) with the service information. Once the application node receives the address, the application node sends an access request to the service node which replies with a simple access code. However, if it cannot locate the service, a name based routing of the message through the fog layer is carried out. The preliminary flow of events

Table 1: Messages and states in LISA protocol.

Message	Description	Options
SDP	Service Discovery	REQ (request), CNF (confirmation)
SRP	Service Registration	REQ (request), CNF (confirmation)
SAP	Service Authentication	REQ (request), CNF (confirmation)
HSP	Handshake	REQ (request), CNF (confirmation)
DRP	Deregistration	REQ (request), CNF (confirmation)
State	**Description**	**Type**
INIT	Initialization	Static
RDY	Ready	Static
SLP	Sleeping	Static
DSC	Discovering	Transient
REG	Registering	Transient
DRG	Deregistering	Transient
ACC	Authenticating	Transient
WHS	Waiting Handshake	Transient
ERR	Error	Transient

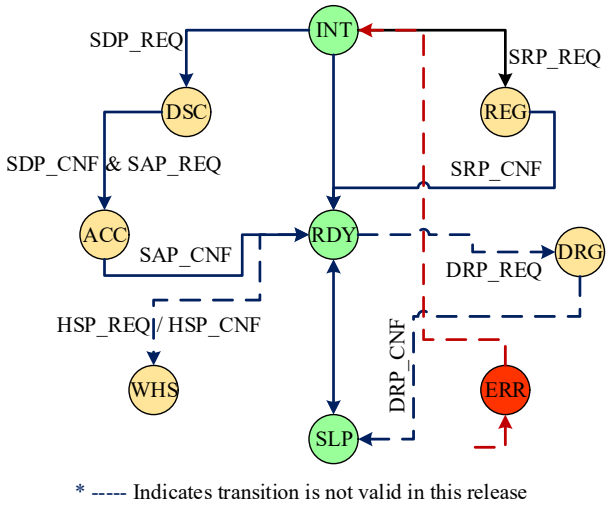

Fig. 6: LISA messages and state transitions

in routing a message is shown in Fig. 8. The details of name based routing are discussed in section 3.3.

The setup messages affect the current state of LISA, and hence the state of the underlying network interface. The network interface and LISA go into a sleep state whenever there is no communication. Advanced synchronization between a service node and application node can be achieved by gathering the registration time and discovery request time from service nodes and application nodes. This feature can provide better power management in the overall subnetwork (Dunkels et al., 2011). The state transition of the bus segment is handled as shown in the Fig. 6. The bus is in transient state if it is in one of the yellow circles (shown in Fig. 6) and can be utilized by the user application only when it is in the ready state. The left path is taken by application nodes, the middle is taken by manager nodes and the right path is for service nodes. There are states which are not fully implemented in the current version (such as ERR and WHS).

4.6 LISA user messages

The user message format for LISA is adapted from NoTA. The service interface describes what parameters are required to use a service and the type of parameters passed. User messages are raw application specific contents exchanged among nodes with a very lightweight header to identify it in LISA. For example, in a health monitoring application domain, where application nodes (sensors) send readings to a service running on another node, the logging service might require the following parameters from an application node: *SensorID* (type unsigned integer), *PatientID* (type unsigned integer) and *Reading* (type float). The type, length (optional) and values of each parameter will be ordered accordingly and copied to the buffer as the body of the message. In LISA, the user message starts with 0x11, which is a code for type of 8 bit unsigned integer (Binnema, 2009), followed by the actual value of *SensorID*. Similarly, *PatientID* and *Reading* are also arranged and inserted into the buffer. The receiving end identifies user message from the header and parse the values according to the service definition exposed for the application node.

4.7 NCN in LISA

An overview of CCN is given in Section 3.3. CCN is aimed at minimizing the problems that arise due to the tight coupling of the current device addressing scheme with the initial purpose of a computer network. In CCN, contents are given unique addresses and this address is

used to route the information over the network. In contrast, Node centric networking gives names for nodes (application, service or manager) which are functional units of an IoT system. It was discussed that LISA enables SOA implementation. Subsequently, the whole system is arranged into groups of clients and services. In addition, one of the main design goals is to provide interoprability of devices. As discussed, interoperability is achieved at multiple layers, one of these layers is semantic interoperability. Semantic interoperability deals with understanding exchanged messages in the right context without ambiguity (Bittner et al., 2005). Due to the huge data coming from the connected devices, which could be redundant over time, has to be analysed in a way that helps to extract information out of it. This semantic information is one of the visions of Internet of Things (Singh et al., 2014). Ontologies are used to describe the common information exchanged among component systems. It is this ontology name of services, that LISA uses to route service requests across protocol boundaries. However, true semantic information out of the data is provided through the cloud and LISA feeds semantically organized data for the upper layer.

The ontology name of services are used only when sending a discovery request message. Ontology names are also hierarchical as in a Resource Description Framework (RDF) identifier. However, only last part of the identifier is used at this stage to identify a node. For internal use by managers, a hierarchical address is given to all nodes in LISA. This is similar to the purpose of the Domain Name System in the Internet. This hierarchical address is generated and managed by manager nodes. Manager nodes map the ontology name to hierarchical address locally as they generate the address. It follows the format:
Domain/Home_Manager/Node_Type/Node_ID. Each part of the name takes 8bits, thus giving a total of 32 bits for each node name. This address is stored inside a table in nodes (this is a similar concept to content stores in CCN). Each of the sections of the name are listed as domain, home, type and node as shown in Fig. 7. This name is unique even across systems in building mash-up of IoT subsystems. This is achieved through the use of unique application domain identifiers for each IoT subsystem. For example, in the motivation example of AAL in section 2, the two major subsystems mentioned (smart home and mobile health system) will have two different domain Id's.

In a discussion on LISA protocol, the specific design of message routing across protocol boundaries was deferred until this point . An application, which is looking for a service, first sends a discovery request to its home

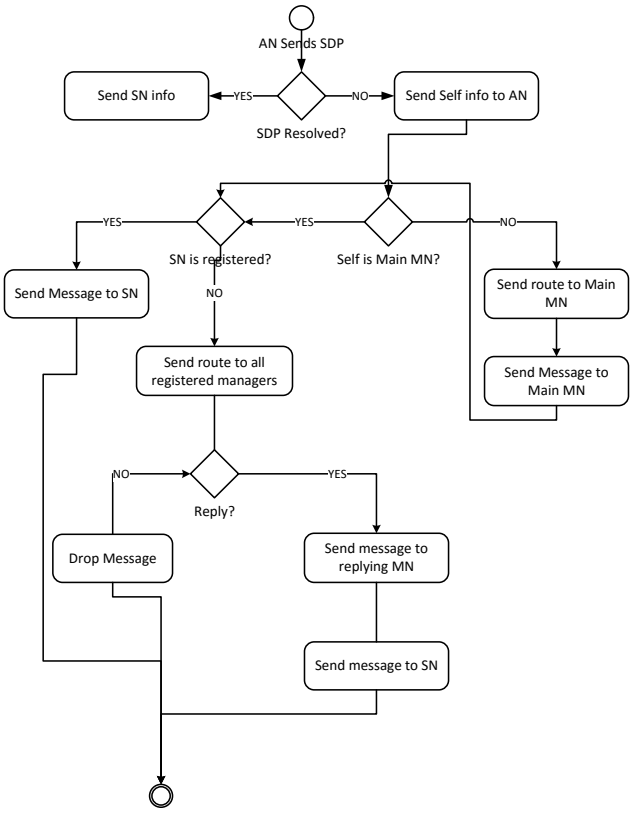

Fig. 8: LISA routing based on custom addressing across subnetworks

manager. The manager looks for the specified ontology name in its local routing table and registers the application with a new address. Ontology name can be similar in two different domains. However, it is unique in a given domain. If the manager cannot find the service, it forwards the message to the main manager. The manager nodes are assumed to have multiple network interfaces and relaxed resource constraints than the other nodes (assumed class 1 devices). The request subsequently goes to the right manager where the service is registered (shown in Fig. 8). In the meantime, the application can send the message to its manager and go to a sleep state. When the service is located, the manager node sends the user message.

5 LISA Demonstrator and Evaluation

To demonstrate the work presented in this paper, two configurations were setup. The first configuration was with only three nodes from each type. The simulation is carried out on a Linux machine running three instances of RIOT operating system on different tap interfaces in native mode. To measure various performance charac-

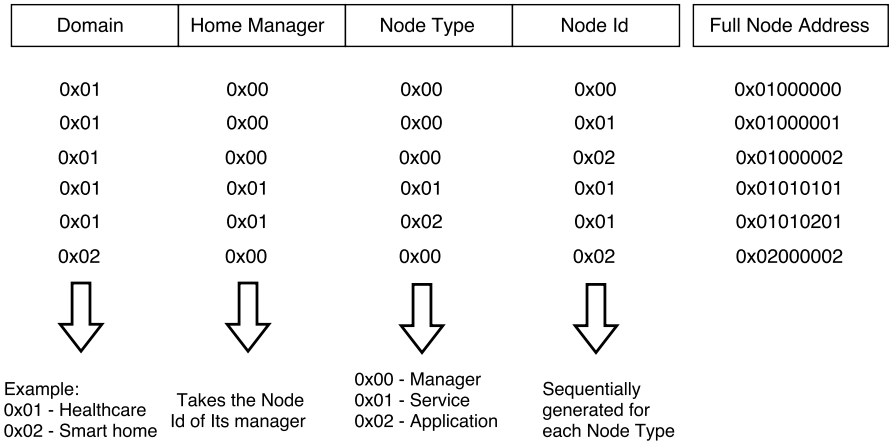

Fig. 7: Node naming in node centric LISA

teristics of the service bus, an interactive user application is built using LISA. RIOT provides a shell module which can be enabled and users can interact with the operating system and the user application. To run the manager node, a shell command **lm -m** is executed in one of the nodes. This initiates the manager application which starts listening to incoming requests and advertises its information to client nodes. The manager node is assumed to run in the Fog layer and has relaxed constraints in terms of power, processing capacity and memory as compared to the end nodes (application and service nodes). Devices running the manager node are also assumed to have multiple network interfaces, such as Bluetooth and Wi-Fi.

The service nodes are composed of intermediate devices with few constraints (devices in class 2). Depending on the functionality, these nodes can have varying up times. Similar to the manager node, the service node is started from the shell using the command **ls -s** followed by the address of the *Home Manager* we want to register to (this is not useful in production), then follows *Service Id* and *Interface Id* (used to locate it by clients). This has been changed in the second demonstrator by replacing the Service Id and Interface Id with the ontology name of the service. The application node also uses similar technique; A shell command **la** followed by the service and interface id it is communicating with, and finally the actual message to transfer.

One of the issues with RIOT when working with the advertisement is the priority of threads. The manager was configured to advertise every five seconds in a separate thread. This is of lower priority than the main thread that handles the incoming requests. However, during the testing phase, the manager was unable to switch from the advertisement thread to the main thread to handle incoming requests. Hence, service nodes and application nodes had to be manually configured with the address of the manager. The first demonstrator was built as a proof of concept for a lightweight version of a service bus for IoT and published in (Negash et al., 2015).

The size of the whole application including the operating system and the required modules is less than 130 Kbytes (as shown in Table 2), from which LISA takes 22 Kbytes (less than 20%). Looking at the running processes from RIOT shell commands, LISA creates only one thread which registers itself for handling incoming packets. This improves power and memory management. Figure 9 shows the sequence of messages in first demonstrator.

Table 2: Size of LISA vs NoTA (Negash et al., 2015).

API name	**Size (KB)**
LISA Only	˜22
LISA with RIOT	˜130
NoTA	˜260

The second part of the demonstrator was an extended version of the first one. Similar to the first demonstration, the simulation is done on a Linux machine running six instances of RIOT in native mode. Two subnetworks are setup each with its own manager, of which one is selected as a main manager. The second manager first gets registered with the main manager and goes into a ready state. Two nodes (one service and one application) register in each manager; a total of six nodes are used in this demonstrator. This creates two subnetworks each simulating a different protocol. An application node in one manager tries to communi-

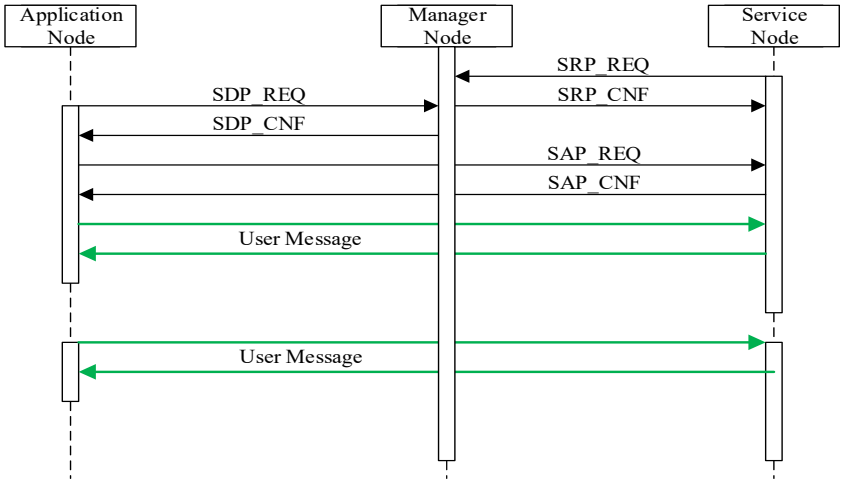

Fig. 9: Sequence of messages in LISA

cate with a service in a different manager. The specific routing algorithm of the service request and application specific message, implemented in this demonstrator is shown in Algorithm 1. The results of the demonstrator are presented in the following section.

5.1 Evaluation

Looking at the operation done for each protocol message, we can analyze the complexity of the related algorithm. For instance, service registration is simply an insertion operation at the manager node. That indicates a complexity of O(1), or a constant time operation. However, discovery of a service by an application node requires variable amount of time depending on the number of already registered services and the data structure used to store the nodes. For this version of LISA a simple list is used to register the nodes. Therefore the complexity of this operation is O(n). To show the performance of LISA, the time required for each protocol message has been monitored. The time required is measured from both sides, the sender (to process confirmation messages or CNF) and receiver (process requests or REQ). The following graphs show the results for different number of trials under the same configuration. When a node is unable to get a confirmation message from the receiver end, LISA tries to send the same message five times before failing to communicate. This naturally leads to longer communication times during the test, as shown in the graphs (Fig. 10). The time taken in a routed message across a sub-network varies depending on where the service node is located. It is basically service discovery request done at multiple manager nodes. To evaluate the cost of using LISA in terms of additional overhead it brings, average times for the different message types are collected.

The simulation is done on a 32 bit Ubuntu 14.04 machine with 4GB of memory and Intel Core 2 Duo processor which has a speed of 3.16 GHz. For a service node to be ready it takes an average of 2.83 µs for Service Registration Request (SRP_REQ) plus 14.67 µs for Service Registration confirmation (SRP_CNF) and one full communication cycle with the manager, which is a total of 17.5 µs delay. Similarly, an application node takes an average of less than 30 µs for discovery and less than 20 µs for authentication in the best case scenario with additional two full communication cycles (one for each process). The total delay for all the setup process is less than 1ms. In addition, there is a bandwidth overhead of 9 bytes as a LISA user message header. Compared to the advantages gained by introducing LISA in the system (interoperability, mobility, ease of programming), the above delays and few additional communication costs with the manager are the design trade-off made in this paper.

6 Related Work

Internet of Things is not a completely new technology but an evolved one from existing ones such as Wireless Sensor Networks and Machine to Machine communication technologies. This ancestral relation also brings the claim that the middlewares developed for prior application requirements can also fit for IoT. There are many

```
Result: Node message delivered to destination
if SDP_Request == True then
    if SDP Resolved then
        Return SDP confirmation;
    else
        Return SELF information;
        Message Received;
        if SELF == Main_Manager then
            while All_Registered_Managers do
                Send route to i^th;
                Send Message to i^th;
            end
        else
            Send route to Main_Manager;
            Send Message to Main_Manager;
        end
    end
else
    Route_Received;
    Message Received;
    if SELF == Main_Manager then
        if Node exists locally then
            Deliver Message;
        else
            while All_Registered_Managers do
                Send route to i^th;
                Send Message to i^th;
            end
        end
    else
        if Node exists locally then
            Deliver Message;
        else
            Drop Message ;
        end
    end
end
```

Algorithm 1: Simplified manager routing algorithm

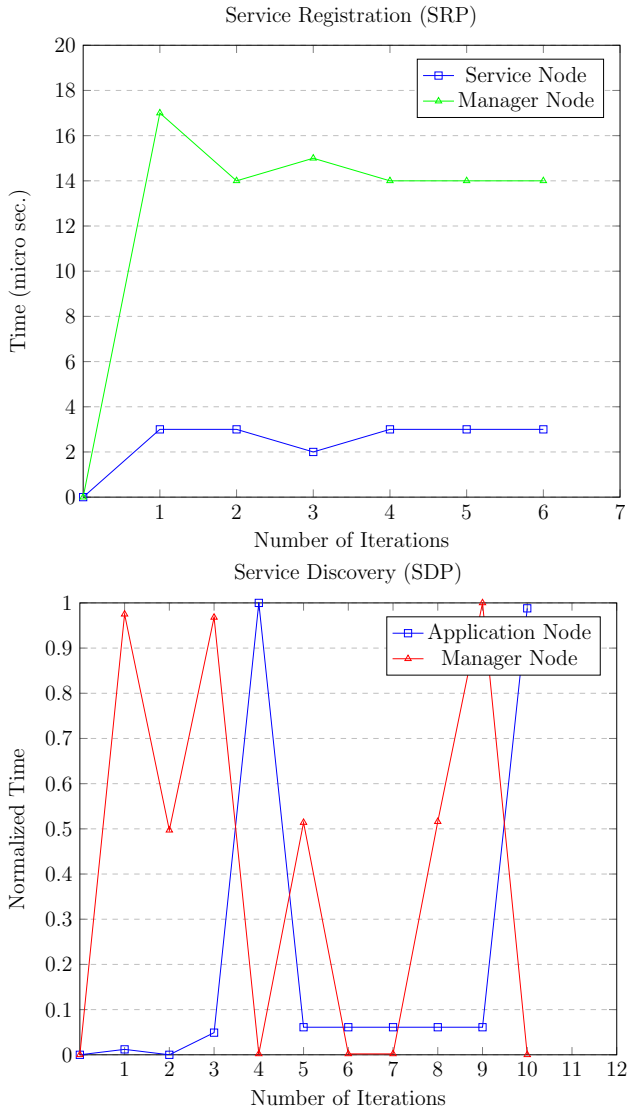

Fig. 10: LISA SRP and SDP processing times

middleware options specially for WSN application (Azzara et al., 2013). Other category of middlewares are designed for data aggregation from various sensor nodes (Perera et al., 2014) without focusing on the heterogeneous communication protocols. A more comprehensive survey of middlewares proposed for IoT are presented in (Razzaque et al., 2016). Several frameworks and design patterns were studied during this work. However, most of the identified frameworks are either heavy for resource constrained devices or those that propose a lightweight middleware run above the application layer protocol which limits the range of supported transport protocols. In comparison, our proposal is closer to low level protocol layers and is lightweight enough for resource constrained devices. The framework introduced by AllSeen is promising in both the applicability and the community behind it, AllSeen Alliance (Allseen Alliance, a), (Allseen Alliance, b). One advantage of the framework is the support of multiple programming languages and familiar platforms in the PC and mobile device domain. However, this work is not focused on the resource-constrained devices which are targeted by LISA. A device service bus presented by Medeiros *et al.* (Medeiros Araújo and Siqueira, 2009) uses web services for interoperability based on Device Profile for Web Services (DPWS). However, similar to the above discussion, this service bus is also targeted for non-resource-constrained nodes.

The initial design of the current Internet was not meant for service centered operation. The work presented by Nordstrom *et al.* (Nordström et al., 2012) addresses the resulting mobility problem accessing the Internet. By introducing a middle layer between the network and transport layer, they provide a means of

addressing services which is network address agnostic. This has been the motivation for LISA to integrate a discovery mechanism which is customized for the IoT domain.

This work is originally started with the inspiration of Network on Terminal Architecture (NoTA) (Binnema, 2009), (Negash et al., 2015). The work introduced in TRIAD (Cheriton and Gritter, 2000) and Information centric networking (Pentikousis et al., 2015) also motivated us to utilize the idea of Node centric networking in LISA. Different types of information centric networking concepts have been implemented so far. One of the main implementations is Content Centric Networking (CCN) (Jacobson et al., 2009). In CCN architecture, contents are given unique names where consumers search for the content using the corresponding name. This requests for specific information known as interest. When a node receives an interest for a specific content, it tries to match the name and if it is the same, the node will reply to the interest. One of the widely used naming scheme in CCN is hierarchical naming. Parts of names indicate chain of relation in the network similar to path names in a file system. For CCN to work, it requires faster and more efficient routing algorithms than routers in traditional network routers, which are also conceptually different. This is due to the fact that high volume of content is routed through a network containing few communicating entities. CCN routers have pending interest store for unanswered interest requests, and content stores to save contents served in previous requests. Data Oriented Network Architecture (DONA) (Koponen et al., 2007) is also a similar approach to CCN with the focus on data. DONA mainly focuses on the interoperability requirements and address mobility, intermittent availability of constrained nodes to move forward to the vision of IoT. In this regard, our work in NCN fits better for IoT in that, it enables a hierarchical, scalable, and service oriented organization of the overall system, and enables interoperability and mobility of IoT components.

7 Conclusions and Future Works

We introduced a lightweight embedded service bus (LISA) to address recent requirements of Internet of Things such as interoperability and mobility, by facilitating the implementation of service oriented architecture. The paper also discussed our implementation of a node centric networking (NCN), a node based routing coupled with LISA. Our implementation has been discussed and demonstrated to be compact for resource constrained devices. LISA offers service discovery, registration and authentication that are essential features to setup and communicate application messages between nodes. Moreover, it presented the benefit of addressing nodes (functional units) in NCN instead of the actual content shared among IoT devices with the additional benefit of extending it for semantic interoperability. The performance of the middleware has also been shown in relation to resource constraints. The middleware takes advantage of Fog computing architecture by implementing federation of nodes, which can route messages independent of the underlying protocol. The performance of the gateways in the Fog layer, which are referred as manager nodes in LISA, is also analyzed and presented. The work is planned to be extended to support multiple operating systems, platform and protocols for IoT with additional layer of semantic interoperability through detailed implementation of domain specific ontologies at the fog layer. The future research works include studying the possibility of running LISA without any operating system, support additional embedded operating systems and adding more features for manager node to activate services. The future work will also contain the development of an optional configuration for quality-of-service messaging depending on the application type. We will release LISA as an open source project thereby providing a community of developers and in house contribution to enhance the features of LISA.

References

A.S. Alghamdi, I. Ahmad, and M. Nasir. Selecting the best alternative SOA service bus for C4I systems using multi-criteria decision making technique. In *International Conference on Computational Technologies in Electrical and Electronics Engineering*, pages 790–795, 2010.

Allseen Alliance. Alljoyn standard core. https://allseenalliance.org/framework/documentation/learn/core/standard-core, a. Accessed: 2015-03-19.

Allseen Alliance. Alljoyn thin core. https://allseenalliance.org/framework/documentation/learn/core/thin-core, b. Accessed: 2015-03-19.

Raffaele Amicis, Giuseppe Conti, Stefano Piffer, and Federico Prandi. Service oriented computing for ambient intelligence to support management of transport infrastructures. *Journal of Ambient Intelligence and Humanized Computing*, 2(3):201–211, 2011. ISSN 1868-5145. URL `http://dx.doi.org/10.1007/s12652-011-0057-z`.

Somaya Arianfar, Pekka Nikander, and Jörg Ott. On Content-centric Router Design and Implications. In *Proceedings of the Re-Architecting the Internet*

Workshop, ReARCH '10, pages 5:1–5:6, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0469-6.

A. Azzara, S. Bocchino, P. Pagano, G. Pellerano, and M. Petracca. Middleware solutions in wsn: The iot oriented approach in the icsi project. In *Software, Telecommunications and Computer Networks (SoftCOM), 2013 21st International Conference on*, pages 1–6, Sept 2013.

Emmanuel Baccelli, Oliver Hahm, Matthias Wählisch, Mesut Gunes, and Thomas Schmidt. RIOT: One OS to Rule Them All in the IoT. Research Report RR-8176, December 2012.

Dirk-Jan C. Binnema. *NoTA programming guide*. Nokia Research Centre, Finland, 2009.

Thomas Bittner, Maureen Donnelly, and Stephan Winter. Ontology and semantic interoperability. In *Large-scale 3D Data Integration Challenges and Opportunities*. CRC Press, 2005. ISBN 978-0-8493-9898-8.

Marjory S. Blumenthal and David D. Clark. Communications Policy in Transition. chapter Rethinking the Design of the Internet: The End-to-end Arguments vs. The Brave New World, pages 91–139. MIT Press, Cambridge, MA, USA, 2001. ISBN 0-262-03292-9.

Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1519-7.

M.R. Bosunia, Anbin Kim, D.P. Jeong, Chanhong Park, and Seong-Ho Jeong. Efficient data delivery based on content-centric networking. In *Big Data and Smart Computing (BIGCOMP), 2014 International Conference on*, pages 300–304, Jan 2014.

M.R. Butt, O. Delgado, and M. Coates. An energy-efficiency assessment of Content Centric Networking (CCN). In *Electrical Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*, pages 1–4, April 2012.

CERP-IOT. Vision and Challenges for realizing the Internet of Things. Technical report, European commission, Information society and media, 3 2010.

David R. Cheriton and Mark Gritter. TRIAD: A Scalable Deployable NAT-based Internet Architecture. Technical report, 2000.

H. Derhamy, J. Eliasson, J. Delsing, and P. Priller. A survey of commercial frameworks for the internet of things. In *Emerging Technologies Factory Automation (ETFA), 2015 IEEE 20th Conference on*, pages 1–8, Sept 2015.

Adam Dunkels, Joakim Eriksson, and Nicolas Tsiftes. Low-power Interoperability for the IPv6-based Internet of Things. Technical report, Wireless Ad-hoc Networks, 5 2011. http://dunkels.com/adam/dunkels11adhoc.pdf.

Kjeld Borch Egevang and Paul Francis. The IP Network Address Translator (NAT). RFC 1631, RFC Editor, May 1994. `http://www.rfc-editor.org/rfc/rfc1631.txt`.

Dave Evans. The Internet of Things How the Next Evolution of the Internet Is Changing Everything. Technical Report 2, Cisco Internet Business Solutions Group (IBSG), 4 2011. Available Online http://www.iotsworldcongress.com/.

Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns, Designing, Building and Deploying Messaging Solutions*. Addison Wesley, Boston, 2003.

IERC AC4. IoT Semantic Interoperability: Research Challenges, Best Practices, Recommendations and Next Steps. Technical report, European Commission Information Society and Media, 8 2013. Available online: http://www.probe-it.eu/.

Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-636-6.

Yaser Jararweh, Mahmoud Al-Ayyoub, Ala' Darabseh, Elhadj Benkhelifa, Mladen Vouk, and Andy Rindos. Sdiot: a software defined based internet of things framework. *Journal of Ambient Intelligence and Humanized Computing*, 6(4):453–461, 2015. URL `http://dx.doi.org/10.1007/s12652-015-0290-y`.

Martin Keen, Amit Acharya, Susan Bishop, Alan Hopkins, Sven Milinski, Chris Nott, Rick Robinson, Jonathan Adams, and Paul Verschueren. Patterns: Implementing an SOA Using an Enterprise Service Bus. Technical report, IBM, 2004. http://www.redbooks.ibm.com.

J. Kiljander, M. Etelapera, J. Takalo-Mattila, and J.-P. Soininen. Opening information of low capacity embedded systems for Smart Spaces. In *Intelligent Solutions in Embedded Systems (WISES), 2010 8th Workshop on*, pages 23–28, July 2010.

Hyojoon Kim and N. Feamster. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119, February 2013. ISSN 0163-6804.

Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A Data-oriented (and Beyond) Network Architecture. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures,*

and Protocols for Computer Communications, SIGCOMM '07, pages 181–192, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-713-1.

D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015. ISSN 0018-9219.

Gustavo Medeiros Araújo and Frank Siqueira. The Device Service Bus: A Solution for Embedded Device Integration Through Web Services. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 185–189, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-166-8.

P. Mockapetris. Domain names - concepts and facilities. STD 13, RFC Editor, November 1987. `http://www.rfc-editor.org/rfc/rfc1034.txt`.

Sergiu Nedevschi, Lucian Popa, Gianluca Iannaccone, Sylvia Ratnasamy, and David Wetherall. Reducing Network Energy Consumption via Sleeping and Rate-adaptation. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 323–336, Berkeley, CA, USA, 2008. USENIX Association. ISBN 111-999-5555-22-1.

Behailu Negash, Amir-Mohammad Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. Lisa: Lightweight internet of things service bus architecture. *Procedia Computer Science*, 52:436 – 443, 2015. ISSN 1877-0509. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

Xuan Nam Nguyen, Damien Saucez, and Thierry Turletti. Providing CCN functionalities over OpenFlow switches. Research report, August 2013. URL `https://hal.inria.fr/hal-00920554`.

Erik Nordström, David Shue, Prem Gopalan, Rob Kiefer, Matvey Arye, Steven Ko, Jennifer Rexford, and Michael J. Freedman. Serval: An End-Host Stack for Service-Centric Networking. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 85–98, San Jose, CA, 2012. USENIX. ISBN 978-931971-92-8.

A. Ooka, S. Atat, K. Inoue, and M. Murata. Design of a high-speed content-centric-networking router using content addressable memory. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 458–463, April 2014.

K. Pentikousis, B. Ohlman, D. Corujo, G. Boggia, G. Tyson, E. Davies, A. Molinaro, and S. Eum. Information-Centric Networking: Baseline Scenarios. RFC 7476, RFC Editor, March 2015.

Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Mosden: An internet of things middleware for resource constrained mobile devices. In *Proceedings of the 2014 47th Hawaii International Conference on System Sciences*, HICSS '14, pages 1053–1062, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-2504-9. doi: 10.1109/HICSS.2014.137. URL `http://dx.doi.org/10.1109/HICSS.2014.137`.

Hauke Petersen, Emmanuel Baccelli, and Matthias Wählisch. Interoperable Services on Constrained Devices in the Internet of Things. In W3C, editor, *W3C Workshop on the Web of Things*, Berlin, Germany, June 2014.

M.A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for internet of things: A survey. *Internet of Things Journal, IEEE*, 3(1):70–95, Feb 2016. ISSN 2327-4662.

Lawrence G. Roberts and Barry D. Wessler. Computer Network Development to Achieve Resource Sharing. In *Proceedings of the May 5-7, 1970, Spring Joint Computer Conference*, AFIPS '70 (Spring), pages 543–549, New York, NY, USA, 1970. ACM.

J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end Arguments in System Design. *ACM Trans. Comput. Syst.*, 2(4):277–288, November 1984. ISSN 0734-2071.

Poornachandra Sarang. *SOA Approach to Integration: XML, Web Services, ESB, and BPEL in Real-world SOA Projects*. Packt Publishing, 2007. ISBN 1904811175, 9781904811176.

D. Singh, G. Tripathi, and A.J. Jara. A survey of internet-of-things: Future vision, architecture, challenges and services. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 287–292, March 2014.

Lu Tan and Neng Wang. Future Internet: The Internet of Things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–376–V5–380, Aug 2010.