Middleware, Operating System and Wireless Sensor Networks for Internet of Things

Ritvik Verma
Department of Electronics and Communication Engineering
M.M.M. University of Technology Gorakhpur, India

Kartikey Srivastava
Electrical and Electronics Engineering Department
S.R.M. University Ghaziabad, India

ABSTRACT
With the evolution of Internet of Things (IoT), various sectors stand at the door of revolution. Recently, there has been a number of proposals for the model of IoT. Despite it, we do not have a reference model where all the key components like supervisory control and data acquisition (SCADA), machine-to-machine (M2M) communication, wireless sensor networks (WSN) and RFID identification is addressed. This paper reviews the architecture, requirements and solutions available for framework, middleware, operating system (OS) and, WSN and MANET, in regard to IoT environment. In addition to this, it highlights the issues and the solutions that can be integrated in the model, like software defined networking (SDN).

Keywords
Internet of Things (IoT); framework; middleware; operating system (OS); software defined networking (SDN); wireless sensor networks (WSNs); MANET.

1. INTRODUCTION
Internet of Things is a network of sensors and actuators having an identification and connected to the internet so as to provide sensing and action as a service. The IoT offers a great opportunity for electronic components manufacturers, Internet service providers and software developers. It is expected by 2022, IoT entities will cross 212 billion, deployed globally and 45% of the whole Internet traffic to be comprised of machine to machine (M2M) communications [13-15].

IoT is set to revolutionize various sectors such as (1) Manufacturing, (2) Agriculture, (3) Healthcare, (4) transportation, (5) Education, (6) Home automation, (7) Automobile and many more. The growing interest in IoT calls for the need of a reference model that takes into consideration factors like heterogeneity, scalability and security amongst others.

This paper contains five sections and provides a brief review of the framework that includes Global Sensors Network, middleware (MOSDEN), operating system (TinyOS) and wireless sensor network. It attempts to briefly review the requirements, solutions available and the challenges that has to be coped up with. Whereas Section II discusses the architecture and its components. Besides this, Section III highlights the requirements of middleware and OS as well as their relation and examples. It discusses the need to integrate software defined networking with IoT. The role and interoperation of Wireless sensor networks and MANETs is presented in section IV. Last Section V includes the summary of the paper.

2. DIFFERENT ARCHITECTURES OF IoT
The IoT should be capable of interconnecting billions or trillions of heterogeneous sensors and actuators through the Internet, therefore it is a critical need for a flexible layered architecture. The ever-increasing number of proposed architectures has not yet converged to a standard model [7]. Meanwhile, there are few models like IoT-A [8] which try to design a common architecture based on the analysis of the needs of researchers and the industry.

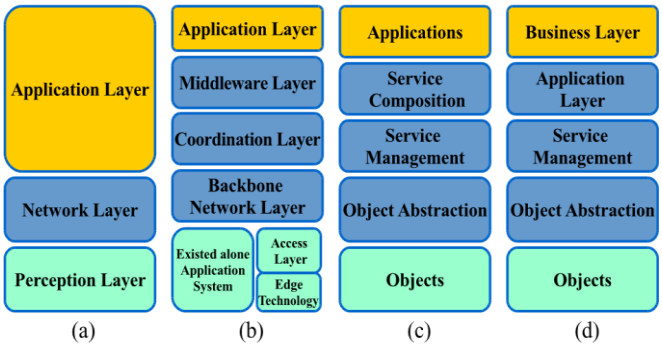

Fig. 1. Internet of Things (IoT) architecture (a) Three-layer. (b) Based-on- Middleware (c) Based-on-SOA (d) Five-layer

2.1 Perception Layer
The first layer, the objects (devices) or perception layer i.e. category 1 as shown in Fig. 2, represents the physical sensors of the IoT that aim to acquire data and process it. This layer includes sensors and actuators to perform different functions. Standards need to be set for plug-and-play mechanisms so as to enable them to configure heterogeneous objects [9]. The perception layer digitizes and transfers data to the Object Abstraction layer through secure channels using protocols like Bluetooth, Zigbee, etc.. RFID and Electronic Product Code (EPC) are essential for identification of the components of this layer[10].

Fig. 2. IoT Device Catagorization on the basis of Computational Capabilities

2.2 Object Abstraction Layer

The Object Abstraction transfers data produced by the sensors (category 3) or the Objects layer to the Service Management layer (category 3) through secure channels. Data can be transferred through various technologies such as ZigBee, 3G, GSM, WiFi, Bluetooth, etc. Furthermore, other functions like cloud computing and data management processes are handled at this designation.

2.3 Service Management Layer

Middleware (pairing) layer or Service Management pairs a service with its requester based on addresses and names. This layer enables the IoT application programmers to work with heterogeneous objects without considering the platform. The server receives the query and then forward it according to the middleware model and routing algorithm. Also, this layer processes received data, makes decisions, and delivers the required services over the network wire protocols [10], [16], [17].

2.4 Application Layer

The application layer provides the services requested by customers. For instance, the application layer can provide temperature and air humidity measurements to the customer who asks for that data [11]. The importance of this layer for the IoT is that it has the ability to provide high-quality smart services to meet customers' needs. The application layer covers numerous vertical markets such as smart home [9], smart building, transportation, industrial automation and smart healthcare [16].

2.5 Business Layer

The management or business layer manages the overall IoT system activities and services. The responsibilities of this layer are to build a business model, graphs, flowcharts, etc. based on the received data from the Application layer. It is also supposed to design, analyse, implement, evaluate, monitor, manage and develop IoT system related elements. The Business Layer makes it possible to support decision-making processes based on data analytics. Management and monitoring of the underlying four layers is achieved at this layer. Moreover, this layer compares the output of each layer with the expected output to enhance services, security and maintain users' privacy [10], [16].

The architectures that corporates network stacks (like the three-layer model) fails to comply with IoT requirements since, e.g., the "Network Layer" does not heed to needs like low power consumption, e.g., low profile connection over 6LoWPAN. More importantly, the layers are supposed to be run on resource-constrained devices while having a layer like "Service Composition" in SOA-based architecture takes rather a big fraction of the time in M2M communications.

3. MIDDLEWARE AND OPERATING SYSTEM

The middleware for IoT acts as a bond joining the heterogeneous domains of applications communicating over heterogeneous interfaces. It is responsible for increasing the level of abstraction for the developers. It often runs on top of OS to providing different functionalities. It runs on category 3, 4 and 5 of Fig. 2 depending on the requirements and the case.

The composite structure of IoT embodies a wide variety of machines ranging from sensors powered by 8-bit microcontrollers to devices powered by advanced processors. Neither conventional UNIX/Windows, nor the existing Real Time Operating Systems are able to meet the demands of heterogeneous IoT applications. We require an operating system that incorporates the essential features for IoT like interoperability, scalability and portability.

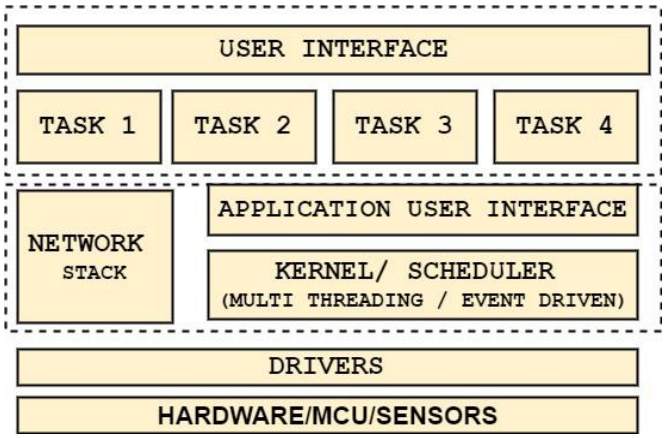

Fig. 3. Generic model of IoT OS

3.1 Requirements of Middleware

The following is a list of major technical challenges that needs to be addressed by middleware solutions for the IoT [32]:

- *Interoperability*: The IoT poses significant amount of challenges for the middleware initiatives since dissimilar devices are required to link and communicate with one another for information exchange. Challenges like these give room for enhanced scope of rigorous research for designing of a middleware that can imbibe majority of the diverse pool of devices within its coverage and provide room for expansion into the domain of devices yet to come in market. [3] Presents an approach in which a sensor based on IEEE 1451 standards is used but presents a significant drawback as it cannot integrate a variety of sensors within itself. In this regard the semantic web approaches as the ones presented in [4] have an edge

over IoTs since interoperability is a constructive feature of the former.

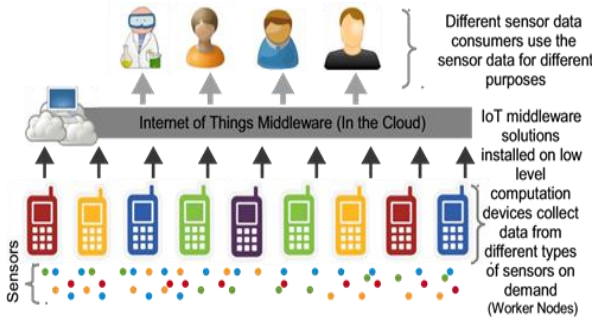

Fig. 4. Middleware Deployment at Worker Nodes and Cloud

- *Scalability*: It counts as one of the major issues to be tackled when it comes to IoTs since these solutions have to support and communicate effectively with a large number of devices. This challenge emerges as a result of having thousands of devices interacting with it, but fortunately, almost in one place. A reliable IoT middleware is required to effectively manage scalability issues so as to enable the operation of the basic functions irrespective of the expanse of the environments [5]. Scalability happens to be one of the drawbacks of the proposed approach in [4], but a pivotal advantage in [3].

- *Abstraction*: Provision of abstraction is a must at various levels for a middleware operating in an IoT environment. It is required for enabling ease in managing heterogeneous input and output hardware devices, hardware and software interfaces, data streams, physicality and the development process.

- *Spontaneous interaction*: These come into picture when new objects enter the wireless range of the other objects already operating in the IOT environment [15]. In such cases middleware is required to manage entities in an arrive-and-operate manner as presented in [6].

- *Flexible Infrastructure*: A heterogeneous environment like the IoT must be capable of distinctly identifying each device operating under it and managing the associated resources without the requirement of additional infrastructure [5]. Using a dedicated server for resource management does not hold in the IoT, because of the high distribution and mobility of devices.

- *Multiplicity*: Two important multiplicity challenges should be dealt with while design an IoT middleware environment. First, the devices that operate under it must communicate with other entities simultaneously [5]. Second, a device participating in an IoT environment is required to select the most suitable services from a massive set of services, because such devices will often rely on services that are available at other nearby devices. They also have to deal with the results returned from different services, which often contradict with one another.

- *Security and Privacy*: Automatic communication of real-life objects poses a huge implication on the concerns of trust, security and privacy. Embedded RFID tags in the personal devices, groceries and even in our clothes can be triggered to respond with their ID and other information. This type of surveillance influences vital aspects of our everyday life. The assurance and support of security and privacy has to be taken into account as a major feature of the middleware design for an IoT [18]. In SOA-based operations, the functions related to security and privacy can be either be constructed on a single layer or distributed among all other layers. In the latter case, other issues have to be considered, so as not to affect the system's performance or introduce excessive overhead.

- *Resource Discovery and Management*: The heterogeneous devices and heterogeneous resources need to be discovered and managed. This needs to be automated too. Middleware is required to discover, identify and managed to maintain an acceptable Quality of Service (QoS).

3.2 Requirements of OS

- *Architecture*: Kernel architecture can be structured in either monolithic, layered or modular microkernel manner. Small memory footprint and less expensive modular interaction are the chief features the monolithic kernel. Better performance is ensured in this case as the control is not exchanged between kernel and microkernel as in the case of microkernel architecture. However, in terms of sustainability, the modular architecture proves better than monolithic, as in the event of failure of a module in the former doesn't result in whole system crash. Also that newer modules can be easily loaded into the memory without disturbing the structure of the system makes the modular architecture a favorable choice. Reliability of the system can easily be ensured while configuration of the modules in case of modular microkernel architecture. Another problem associated with the monolithic kernel is- as the kernel code becomes long and complex, it becomes harder to understand and hence, configure. The third approach of layered architecture is less modular as compared to microkernel. But it is more manageable, reliable and less complex than monolithic kernel.

- *Programming Model:* Out of the many factors that determine the structure of the programming model, parallelism, memory hierarchy and concurrency can be considered as the chief decisive factors. The resultant programming model affects the system performance, productivity, reliability, security and flexibility. The architecture of the programming model is aimed at optimum utilization of the underneath architecture for the applications running on top. The programming model also focuses on increasing the developer productivity. The role of the programming language is to abstract this underlying system The APIs and programming languages implement a programming model and abstracts underlying system. Therefore, the programming model should equip the programmer with the liberty to make the best possible of the system architecture. Assembly language is the best alternative to interface with the hardware but support to high level languages is required for easy development.

- *Scheduling*: The scheduling algorithm plays a decisive role in determining the latency (turnaround time, response time), throughput, fairness and wasting time as a result of which the whole scheduling strategy is

affected. Given the variety of the IoT tasks, there are many applications with strict time constraints. In order to be capable of meeting the deadlines in real, the scheduler must operate in real time so that tasks are accomplished within the allotted time span. Moreover, the schedulers should be energy efficient and multitasking in IoT systems.

- *Networking:* Internet With internet connectivity being one of the fundamental requirements of the IoT environment, such devices should be capable of operating with low power consumption. Conventional TCP/IP stacks and WSN networking technologies are not suitable for IoT. While former fails to achieve the goals of less complexity, less memory and low power, latter needs intermediate proxies to enable different communication platforms to converse with peers. The individual requirements of smart systems can be fulfilled by WSN protocols like ZigBee, Bluetooth, Z-Wave, Wavenis and many more as such yet these platforms fail to suit the large scale communication requirements of IoT [19]. An open standard that allows seamless communication over the internet is required. Apart from it, the IoT stack should be light-weight, reliable and Internet-enabled. Flexibility of the stack is also a chief area of concern as it enables the configuration of a large number of devices with minimal changes in it. The support to Ipv6 is mandatory in IoT systems to have unique identities in tremendously large networks. Mechanisms like 6LoWPAN (Low-power Wireless Personal Area Network over IPV6), RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) and CoAP (Constrained Application Protocol) are designed for low-power systems. Header compression and inclusion of minimal features help in keeping the protocols viable for IoT.

- *Memory Management:* Memory management provides an abstraction to programming taking care of things By performing tasks like caching, memory allocation/de-allocation, virtual memory, logical-physical address mapping, memory protection at the back-end the memory management module abstracts the programming structure. In IoT devices, where simple and small kernel is a primary goal, many IoT OSes do not have Memory Management Unit (MMU) and Floating Point Unit. The application type and the support of the underlying platform also plays an important role in the performance of the memory management functions. The memory allocation can either be static or dynamic. The static memory allocation is simpler but flexibility of run-time memory acquirement can be ensured by the dynamic approach.

3.3 Relation between Middleware and OS

High end computational devices, low end computational devices and sink nodes have operating system with comprehensive network stack running on them to communicate within a local network or through internet using IPv6. For easing the development of applications, level of abstraction needs to be increased at all levels. Depending on the level in the hierarchy we have different middleware available to us.

TinyDB is a query-based middleware based on TinyOS with good data management and power efficiency but with poor middleware functionality [20, 21]. TinyCubus is another middleware based on TinyOS which is cross layer, generic and flexible and that can manage new application requirements [25].

We do not have a versatile solution for all layer. So, middlewares at different levels need to interact efficiently and reliably. One of the combination is MOSDEN (node-level middleware) [26] running on top of Android and interacting with Global Sensors Network (GSN, Cloud IoT middleware). Application development is plugin based and communication is REST based peer-to-peer over HTTP. Sensors are registered with GSN and the data can be accessed using the platform.

Other solutions are TWINE [22], Ninja Blocks [23] and Smart Things [24] amongst others. Having their own proprietary software installed on the nodes, these do not provide an opportunity for development.

3.4 Examples of Middleware and OS

Few examples of OS are discussed here. Contiki [27] is a portable and flexible OS implementing a hybrid protothread model, which supports both event-based and multi-threading. RIOT [28] has been developed in two parts, hardware dependent and hardware independent. Hardware specific part is configured to implement the solution in C. TinyOS [20] is an OS based on components (software modules around hardware). It is written in NesC. Hardware access is simple and it implements Berkeley Low Power Internet Stack (BLIP). Other OS options are LiteOS [30], FreeRTOS [29], and OpenTag [31] amongst others.

There is a diverse range of middleware solutions depending on the layer and level of abstraction (e.g. local or node level) and implementation domains (e.g. WSN, RFID, etc.). There are various types of middleware [32] based on the nature of operation like event-based, service-based, VM-based, agent-based, tuple-based, database-based and application specific.

Other solutions for middleware are RUNES (event-based), TinySOA (service oriented), Maté (VM based), COUGAR (database oriented) and TinyLIME (tuple-space).

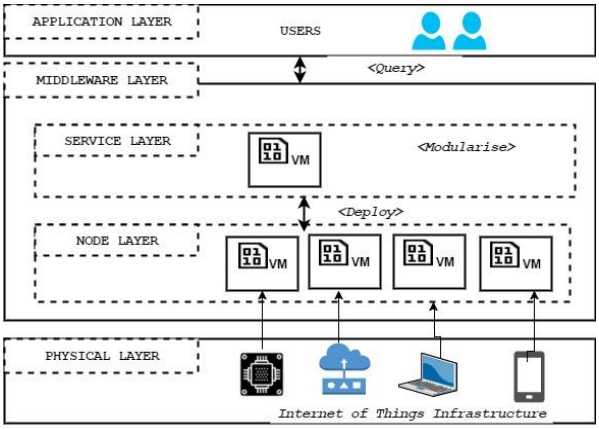

Fig. 5. Model of VM-Based Middleware

3.5 Software defined Networking

The steep rise in the number of connected devices to the internet due to its service model and M2M communication is going to result in overloading and paralysis of the network. Lack of single protocol standards add on to the complex control problems. Software defined networking aims to solve the problem. It reinforces the network, increases bandwidth, reduce latency and linkup time. There are numerous merits of SDN that includes, efficient control of traffic, effective use of resource, ease in data acquisition, reliable data analysis, traffic pattern analysis (debugging tool).

Some of the available solutions for SDN are OpenRoad, Oclin, OpenRadio, OpenRAN, CellSDN and SoftMoW. One of the popular solution is OpenFlow [33], which can be enabled in switches and other products. Every incoming flow is evaluated independently and the respective task is performed in accordance with the matching flow, following the unified control protocol.

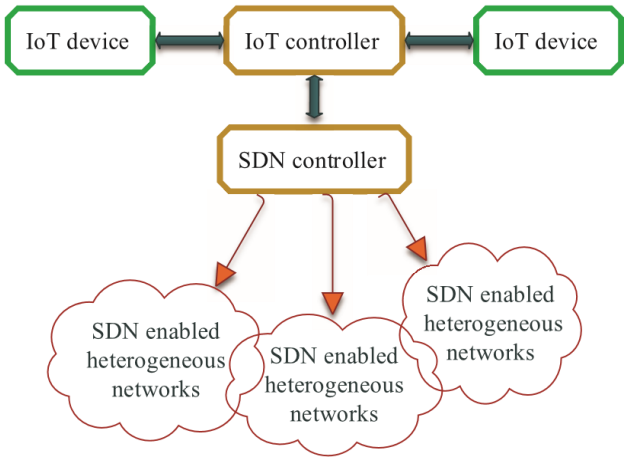

Fig. 6. SDN – IoT Integration Architecture

SDN can be implemented by implementing a OpenFlow-like protocol in IP layer. One node acts as controllerwhile others are controlled device. Virtual Machine (VM) can be used to simulate the network behaviour with OpenFlow. SDN enabled hybrid network implemented on NetFPGA reduces latency and linkup time, where the control circuit and packet switching is governed by unified control protocol.

4. WIRELESS SENSOR NETWORKS AND MANET

4.1 Wireless Sensor Networks (WSNs)

A Wireless Sensor Network consists of spatially distributed sensor to monitor physical or environmental conditions such as temperature, sound, vibration, pressure, motion or pollution and to cooperatively pass their data through network to a main location [34]. Applications of WSN includes healthcare, industrial automation, indoor/outdoor environmental monitoring, inventory location awareness etc.

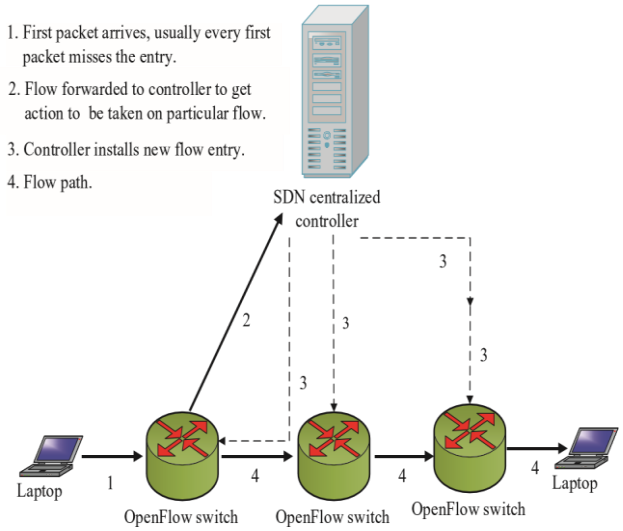

Fig. 7. SDN OpenFlow Switching

Today, the aim is to build on the already existing sensor network, so as to be able to exploit the available resources and, to control actuators and acquire data using sensor, remotely over the internet. There are two types of wireless sensor network: (1) which can communicate over IP protocol. (2) which cannot communicate over it. 6LoWPAN/IPv6 offers IP solutions while Zigbee, Z-Wave, Wavenis and Insteon offers non-IP solutions.

LoWPAN [35] is made up of LoWPAN. Edge router plays an important role as it routes traffic in and out of LoWPAN. Each LoWPAN node has a unique IPv6 address. 6LoWPAN does not require an infrastructure to operate and can operate as an ad hoc LoWPAN.

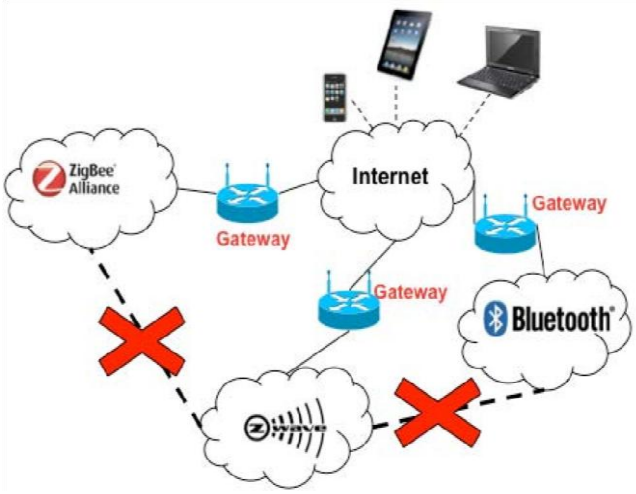

Fig. 8. Interaction between Heterogenous WSN

Using IP protocol for WSN offers us the opportunity to exploit web services. Constrained Application Protocol (CoAP) has been developed for deployment in IoT environment. Its effective for resource constrained setup [35]. For IoT and M2M, HTTP is too bulky with high power and bandwidth consumption. It can run on any device which is User Datagram Protocol (UDP) or its equivalent compatible. The protocol has been provided with standardized framework by the joint collaboration Internet Engineering Task Force (IETF) and CoRE group and is still evolving. CoAP needs to consider optimizing length of datagram and satisfying REST protocol to support URI (Uniform Resource Identifier). It also needs to provide dependable communication based on UDP protocol. CoAP features are as follows [2]:

- Constrained Web Protocol fulfilling M2M requirements.
- Security binding to Datagram Transport Layer Security (DRLS).
- Asynchronous message exchanges.
- Lower header overhead and parsing complexity.
- Uniform Resource Identifier (URI) and Content –type support.
- Simple proxy and caching capabilities.
- UDP binding with optimal reliability supporting unicast and multicast requests.
- A stateless HTTP mapping. Allowing proxies to be built proving access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.

REST consists of coordinated set of guidelines and rules for building effective application architecture. The HTTP transfer

protocol lies over the RESTful architecture [36]. Under the HTTP protocol, the client-server communicates via request-response over the internet. The client could then have its own logic to present the response to format the communicated data. Contrary to the standardized SOAP architecture that strictly relies on the provided documentation, the RESTful architecture is quite flexible. The REST client-servers can communicate using common formats like XML, JSON, text etc. The messages can be exchanged in most of the HTTP methods. Also, the REST HTTP protocol does not solely depend on a strict service definition. The important constraints which is followed by the HTTP transfer protocol in accordance to REST include cacheable responses, uniform client-server interaction, statelessness where no client information is stored during request on the server, layered system and uniform interface.

4.2 MANET

MANET stands for Mobile Ad hoc Network. In this network, several mobile nodes are present in the pool which can independently interact with each other wirelessly. It does not follow a strict top-down architecture. Each device becomes an inherent part of the network infrastructure. The need for central communication is obviated since each device can freely communicate with other devices in the pool. Some of the features of Mobile Ad hoc Networks include low latency, real time position location, secure transfer, high bandwidth and supports over the air-rekeying.

In a MANET driven system, there are several nodes interconnected. At any time, several mobile devices can join or leave the network pool without hindering the efficiency of data transmission. The data to be transmitted can be routed through any of the unique passages connecting different nodes. This makes the network reliant, robust and resilient.

MANET can be further divided into following categories depending on the area of deployment:

- Military or Tactical MANET
- Smart Phones ad hoc Network (SPANs)
- Vehicular ad hoc Network (VANET)
- Internet based Mobile ad hoc Network (iMANET)

Their easy deployments followed with high bandwidth efficiency, lower network overhead, lower energy consumption and better reachability stand as a proof for the underlying potential in this technology.

4.3 Integration of WSN and MANET

While WSN is a network of devices, MANET is a network of people. Their convergence greatly improves data acquisition and digital record of physical environment. The idea is to create dynamic cluster of network according to a protocol. This arrangement offers flexibility, e.g., if data is marked urgent, WSN-MANET integration support can be dynamically activated for speedy execution of the task, which would otherwise be kept inactive to keep power consumption low. It is cost effective and also result in efficient network. There usually is a tradeoff between coverage range, data rate and routing support.

For physical layer Bluetooth Low Power (BLE) and Zigbee is suitable. While IPv6 Routing Protocol for Low-power and lossy networks (RPL) is suitable for networking layer and can operate over any physical layer.

5. CONCLUSION

There is growing interest in IoT and it is set to change our lives in all realms. Its architecture has multiple layers employing multiple technologies. Each layer has its own purpose, requirements and challenges. There is need to understand them, innovate and employ technologies to complement one another.

The five-layer architecture is the answer to IoT requirements of protocols for resource-constrained devices. CoAP, MQTT, 6LoWPAN, etc. are essential for the reference model that is surfacing, for IoT.

Middleware acts as an interface between developers and objects. Its increases the level of abstraction. It operates on top of OS performing different functions for different categories (fig. 2). They have functional and non-functional requirements like interoperability, scalability, security and privacy amongst others. The model for middleware is of different types like database-oriented, tuple-space, service-based, etc.

Operating system requirements for IoT solutions is different from traditional OS and real time OS. Scalability, interoperability, scheduling and networking demands are specific for IoT. Some of the available solutions are Contiki, RIOT, FreeRTOS and TinyOS.

Stupendous increase in connected devices threatens to paralyze the internet if an efficient method of controlling the network is not sought. Software Defined Networking (SDN) offers to effectively control the traffic, analyze data and acquire data. OpenFlow is a suitable solution for it.

Wireless sensor networks connected by non-IP protocol has been here for a while now. IoT demands them to be connected to the internet for performing task and acquiring data remotely. 6LoWPAN is a low profile protocol to implement that for resource constrained devices. CoAP following REST criteria exploits the web services.

MANET is a significant part of IoT which is required to acquire data effectively and make decisions soundly. Integrating WSN and SDN is necessary for their effective use in IoT environment.

The challenges that IoT poses like heterogeneity, scalability, flexibility, security and privacy is yet to be surmounted. Amongst all models, a reference model is required to come up and standards need to be set for commercial deployment of more IoT devices.

6. REFERENCES

[1] H. Zhou, The Internet of Things in the Cloud: A Middleware Perspective, 1st ed. Boca Raton, FL, USA: CRC, 2012.

[2] C. Perera, A. B. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the Internet of Things: A survey,” *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 414–454, May 2013.

[3] K. Aberer, M. Hauswirth, A. Salehi, “Middleware support for the Internet of Things,” In: 5th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze, Berlin, Germany, Sep. 2006.

[4] A. Gómez-Goiri, D. López-de-Ipiña. “A Triple Space-Based Semantic Distributed Middleware for Internet of Things,” In LNCS Vol. 6385, pp. 447-458. Springer, July 2010.

[5] F. Mattern, C. Flörkemeier. From the Internet of Computers to the Internet of Things. Informatik-Spektrum, 33(2), 2010.

[6] K. Paridel, E. Bainomugisha, Y. Vanrompay, Y. Berbers, and W.D. Meuter, "Middleware for the Internet of Things, Design Goals and Challenges", ECEASST Journal, ISSN 1863-2122, 2010.

[7] S. Krco, B. Pokric, and F. Carrez, "Designing IoT architecture(s): A European perspective," in *Proc. IEEE WF-IoT*, 2014, pp. 79–84.

[8] EU FP7 Internet of Things Architecture Project, Sep. 18, 2014. [Online]. Available: http://www.iot-a.eu/public.

[9] Z. Yang *et al.*, "Study and application on the architecture and key technologies for IOT," in *Proc. ICMT*, 2011, pp. 747–751.

[10] M. Wu, T. J. Lu, F. Y. Ling, J. Sun, and H. Y. Du, "Research on the architecture of Internet of Things," in *Proc. 3rd ICACTE*, 2010 , pp. V5-484–V5-487.

[11] L. Tan and N. Wang, "Future Internet: The Internet of Things," in *Proc. 3rd ICACTE*, 2010, pp. V5-376–V5-380.

[12] M. A. Chaqfeh and N. Mohamed, "Challenges in middleware solutions for the Internet of Things," in *Proc. Int. Conf. CTS*, 2012, pp. 21–26.

[13] D. Evans, "The Internet of things: How the next evolution of the Internet is changing everything," CISCO, San Jose, CA, USA, White Paper, 2011.

[14] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Anal. Future*, vol. 2007, pp. 1–16, Dec. 2012.

[15] S. Taylor, "The next generation of the Internet revolutionizing the way we work, live, play, and learn," CISCO, San Francisco, CA, USA, CISCO Point of View, 2013.

[16] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future Internet: The Internet of Things architecture, possible applications and key challenges," in *Proc. 10th Int. Conf. FIT*, 2012, pp. 257–260.

[17] M. A. Chaqfeh and N. Mohamed, "Challenges in middleware solutions for the Internet of Things," in *Proc. Int. Conf. CTS*, 2012, pp. 21–26.

[18] L. Atzori, A. Iera and G. Morabito, "The Internet of Things: A Survey", Computer Networks, 54(15): 2787-2805, 2010.

[19] H. Will, K. Schleiser, and J. Schiller. *A Real-Time Kernel for Wireless Sensor Networks Employed in Rescue Scenarios*. In Workshop on Practical Issues in Building Sensor Network Applications (SenseApp), 2009.

[20] *TinyOS* [Online]. Available: www.tinyos.net/

[21] *TinyDB* [Online]. Available: tinydb.readthedocs.io/en/latest/extend.html

[22] *TWINE* [Online]. Available: supermechanical.com

[23] *NinjavBlocks* [Online]. Available: ninjablocks.com

[24] *Smart Things* [Online]. Available: smartthings.com

[25] P.J. Marrón, A. Lachenmann, and D. Minder, "TinyCubus: A Flexible and Adaptive Framework for Sensor Networks," in 2nd European Workshop on Wireless Sensor Networks, 2005, pp. 278-289.

[26] C. Perera, P. P. Jayaraman, A. Zaslavsky, P. Christen and D. Georgakopoulos, "MOSDEN: An Internet of Things middleware for resource constrained mobile devices", *Proc. 47th Hawaii Int. Conf. Syst. Sci. (HICSS)*, pp. 1053-1062, 2014.

[27] *Contiki* [Online]. Available: www.contiki-os.org.

[28] *RIOT* [Online]. Available: www.riot-os.org.

[29] *FreeRTOS* [Online]. Available: www.freertos.org.

[30] *LiteOS* [Online]. Available: http://lanterns.eecs.utk.edu/software/liteos/

[31] *OpenTag*[Online].Available: www.indigresso.com/wiki/doku.php?id=opentag:main.

[32] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade and Siobhan Clarke, "Middleware for Internet of Things: A Survey", *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70-95 .

[33] *OpenFlow* [Online]. Available: www.opennetworking.org/sdn-resources/openflow/

[34] *Wireless Sensor Networks* [Online]. Available: www.scribd.com/doc/50634760/MANET-vs-WSN

[35] L. Mainetti, L. Patrono, and A. Vilei, "Evolution of wireless sensor networks towards the Internet of Things: A survey". Proceedings of the 19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2011). Split, Croatia, 2011, pp. 1-6.

[36] Fielding, Roy Thomas (2000). Chapter 5: Representational State Transfer (REST).