Available online www.jocpr.com

Journal of Chemical and Pharmaceutical Research, 2014, 6(6):2927-2933

Research Article

ISSN : 0975-7384
CODEN(USA) : JCPRC5

Secure data aggregation in mobile sink wireless sensor networks

Kong Xiangsheng and Chang Qing

Department of Computer & Information, Xinxiang University, Xinxiang, China

---

ABSTRACT

In this paper, concepts for adapting the IPFIX protocol to the needs of wireless sensor networks have been investigated, resulting in the development of the protocol TinyIPFIX, which is an adaptation of the IP Flow Information Export (IPFIX) protocol. The new protocol has been assessed in a representative use case involving a building application. TinyIPFIX has been extended by compression capabilities and by aggregation functionality. Furthermore, extensions to support secure data transmission have been developed, using the protocol Datagram Transport Layer Security (DTLS). This solution ensures that data collected by sensor nodes is transmitted via secure channels to a global data sink, and that authorized access is ensured from a data sink to a wireless sensor network.

Keywords: WSN, TinyOS, ZigBee

INTRODUCTION

As previously mentioned, it is required to integrate wireless sensor networks into the Internet of Things. It is assumed that sensor devices have enough resources to support IP communication. An overview of IP solutions for sensor devices is presented, e.g. ZigBee, IPv6 over Low power Wireless Personal Area Network (6LoWPAN), and the Berkeley Low-power IP stack (BLIP) representing the underlying stack. Due to the limited resources of sensor hardware, especially energy, memory, and computational capacity, supported solutions must be developed in a resource efficient way.

In order to support interoperability of protocols and heterogeneity in the hardware, the developed solutions must be flexible and independent of the underlying stack (e.g. 6LoWPAN, BLIP), which is responsible for the network and transport functionalities. As a consequence the integration of protocols and algorithms on the application layer with focus on standardization is preferred as solution. The application layer is located separately above a complex (called network stack) that includes all underlying layers (e.g. transport, network, physical). The protocols and algorithms in the application layer communicate with the underlying layers via interfaces. For example, if a protocol using 6LoWPAN on the underlying layer is developed and ZigBee is now exchanged by BLIP, it is only required to modify the interfaces between the application layer and the underlying stack but not the application itself. Instead of developing new isolated solutions, this dissertation focuses on the analysis of existing standards, for example of IP networks, and the possibility for application and protocol transfer to constrained hardware, such as sensor nodes, and transfers advantageous characteristics (e.g. message format, energy efficiency) into one combined protocol - TinyIPFIX and its extensions.

RELATED WORK

In general, a wireless sensor network consists of different sensor nodes (white) from different vendors which collect individual data and transport it to one or more sink(s) (black) as illustrated in Figure 1. The number of participating nodes depends on the network size and may cause a long communication link to the sink with several hops in between. Depending on the placement of the sensor nodes, some nodes are not able to communicate with the sink directly. To overcome this, the network must support routing functionality in order to route data packets to the sink in an efficient way. Therefore, the routing algorithms take the current network status into account in order to be able to calculate the

optimal route towards the sink. For example, depending on the location of sensor node A it would be expected that the routing algorithm would prefer to forward the collected data to sink 2 and further to the global data sink. Instead, the underlying routing algorithm (e.g. BLIP) decides to route the data over five hops to sink 1 using direct addressing of the sink by its individual IP address. From sink 1 the data is forwarded using LAN connections to the global data sink. In both cases the data would arrive the global data sink but the required time might be different [1].

The sink is a sensor node with gateway functionality (e.g. IP Basestation1). It forwards the wireless received data to a wired infrastructure, such as a server or PC, which is responsible for the further handling of the data (e.g. interpretation and analysis of the data, forwarding to the global data sink). The terms sink and base station are equivalents and together with a server connection a gateway is formed. Various functions are performed on the server such as data storage, data pre-processing, visualization or node configuration. In general, the gateway has a connection to other components of the cyber-physical system and provides the collected data to the components for different application purposes (e.g. online analysis, entity management).

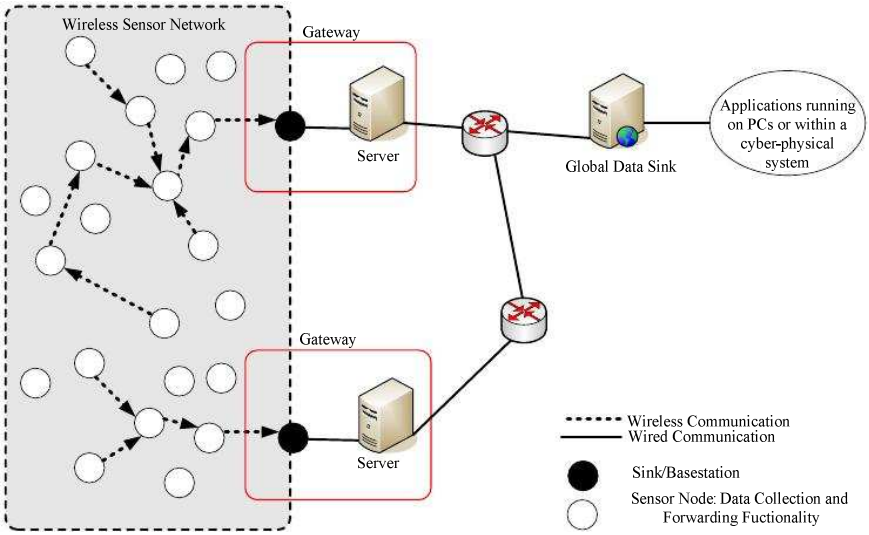

Figure 1. General illustration of a wireless sensor network

The research questions answered in this dissertation focus on efficient data transmission in wireless sensor networks with support of a security solution. Security considerations are required, because of the integration of wireless sensor networks into the Internet of Things and because of the connection between data and personal/sensible information (e.g. GPS). In order to develop a resource efficient solution the requirements of wireless sensor networks are kept in mind. The developed solution should support different hardware platforms and allow integration of new features into a modular architecture. This dissertation deals with the following research questions in the field of efficient data transmission:
(E1): Is the IP Flow Information Export (IPFIX) protocol a viable solution for transmission of sensor data in wireless sensor networks?
(E2): Is it possible to combine data pre-processing techniques (e.g. aggregation) with the IPFIX protocol within the network?
(E3): Can all sensor platforms perform TinyIPFIX and its extensions (compression, aggregation), as well as TinyDTLS?

Due to the integration of wireless sensor networks into the Internet of Things and the relation between data and private/sensible information a secure data transmission is very important. Today, applications of wireless sensor networks prefer to use the unreliable User Datagram Protocol (UDP) instead of the reliable Transmission Control Protocol (TCP). As described by Wagenknecht et al., deployed wireless sensor networks use UDP for data transmission towards a sink and TCP is used for administrative functions (e.g. node configuration). If TCP would be used, the standard TLS protocol could be used. It requires that both communication endpoints support the same network layer. Another problem for using TLS over UDP is the missing authentication of packets if packet loss occurs which is possible when UDP is used.

In order to support secure data transmission between participating sensor nodes, this dissertation deals with the following research questions in the field of security:
(S1) Is it possible to secure data transmission in wireless sensor networks with known standards from IP networks?

(S2) Can DTLS be performed on severely resource constrained hardware as used in wireless sensor networks?

DESIGN PRINCIPLES

The operators of wireless sensor network set a high standard to good support of quality of service, energy efficiency, and scalability of the performed functionalities. In order to accomplish the operators' requests, design principles are important as described by S. Fouladgar et al. in reference [2]. The design principles introduced below, are differently weighted according to the chosen application scenario, the performed functionalities, the network size, and the resources of the hardware used.

Hardware Specification of Sensor Devices

The standard equipment of a sensor node consists of a microcontroller, a memory unit, a communication device (radio), a power supply, and one or more sensors or actuators. Depending on the hardware design other components might be included such as LEDs, power switches, external power connectors, external RF connectors or expansion connectors. The main task of sensor nodes is to collect data and forward it. Depending on the application some sensor nodes can perform additional tasks, such as packet aggregation or data pre-processing, within the wireless sensor network.

The most relevant component of a sensor node is the microcontroller. It has several responsibilities such as data collection from sensors or actuators, data processing, decision management, and flow control. The memory is subdivided into a random access memory (RAM) and the read-only memory (ROM), which is sometimes an electrically erasable programmable ROM (EEPROM) or a flash memory. In contrast to ROM, RAM looses its content by power loss. Thus, it is used for buffering purposes, especially for data that can be changed while a program runs. The executing code is stored in the ROM in order to avoid reprogramming after power loss. The node, therefore, must only be programmed once and can be reused several times until a code update is required.

The communication device is responsible for the communication ability of the sensor node as described in detail in reference [3]. Depending on the application scenario the number of sensors and actuators differ. The power supply must be dimensioned according to the hardware, the application scenario, and the targeted lifetime of the system.

Energy Saving Methods

Usually, the most energy consuming procedure is the transmission of data to the next hop towards a sink followed by decoding and processing operations directly on the node. In order to ensure a longer lifetime of the wireless sensor network energy saving techniques should be integrated into the system.

One idea is to reduce energy consumption by implementing different modes of activity for the nodes as described in reference [4]: full active, idle, and sleep. Most power is consumed when the node is fully active, which means everything requires full power for listening, sending, and data collecting. Sensor nodes in the sleep mode consume very little energy-microjoule instead of millijoule [5]. Normally, nodes are programmed with internal clocks that wake them up in predefined intervals in order to perform, e.g. data collection followed by sending data directly to the next hop afterwards, before they fall into sleep mode again. The idle mode is a mode in between those two modes. Here the node actively listens to the surrounding traffic for beacons that let it know when to wake up and when to perform operations.

Another idea to reduce energy consumption is software based and focuses on message size and network traffic. It has been proven that messages with smaller size consume less transmission energy than bigger messages [6]. Therefore, one strategy is the reduction of overhead. In the context of sensor networks overhead is the meta information connected to each measurement, which is anytime the same for the same sensor node. Thus, splitting of sensor data packages into a message that includes meta information, and a message with the measured values is the strategy of choice. The message including meta information is sent out to all network components when the sensor node boots. After this announcement the sensor node only transmits messages, including the measured data, and refers to the before announced meta information. Also, each packet should produce low overhead in order to offer more space for individual payload, including relevant data. This approach can be realized by performing compression techniques on the messages components (e.g. headers). Another strategy is the reduction of traffic within the network itself in order to save energy. In this case either message aggregation or pre-processing of data can be performed on nodes within the network. The simple idea behind this approach is that sometimes not all data is required in order to perform a specific action. For example, only the average room temperature is required to manage the cooling system in a room. A complete different method to raise the system's life time is to charge the power resources by using solar panels or environmental inputs such as vibration or temperature.

OPERATING SYSTEM FOR SENSOR DEVICES

Today the operating systems TinyOS and Contiki are very popular for wireless sensor networks which are briefly characterized in Section 2. A few years ago researchers started to develop new operating systems based on Unix (e.g. MantisOS, LiteOS), but those approaches are not as popular as TinyOS and Contiki and do not support the hardware. Thus, they are not addressed in this section. Finally, a brief look on code porting possibility between TinyOS and Contiki is presented, which is driven by industry partners and the ongoing development in the Internet of Things [7].

Characteristics of TinyOS

TinyOS is a research driven operating system, which is used for hardware with limited resources such as Berkeley Motes (e.g. MICA2, MICA2dot, IRIS, TelosB) that are used in the deployed wireless sensor network in this dissertation and were introduced in Section 2. TinyOS is an open source project. It was developed by David Culler and Jason Hill at the University of California, Berkeley, USA in 2000 especially for wireless sensor networks based on the requirements of Berkeley Motes. TinyOS is a component-based operating system and has an efficient multi-threading engine, which is composed of a two-level-scheduler and realizes the computer-time-spreading for threads. The application code consists of a Makefile including compiling commands, module files including configuration information, and one or more configuration files including the required information of the interfaces used and the component wiring [8]. The following two sources of concurrency are necessary prerequisites in order to understand the execution order of the two-level-scheduler in TinyOS:

Tasks, which contain current network routing and data preparation, typically take longer to finish, because hardware events have higher priority.

Events on the other hand must be handled immediately, so that long duration blockades caused by current applications and data loss are prevented.

Characteristics of Contiki

In contrast to TinyOS the operating system Contiki is an industry driven operating system. In 2004 the development of Contiki by Adam Dunkels at the Swedish Institute of Computer Science was driven by the request of a light weight operating system (2 kB RAM and 40 kB ROM) for embedded systems that later on was adjusted to the requirements of wireless sensor networks. The technical term embedded system summarizes small and limited hardware that is included in big systems (e.g. control system in a washing machine). Those embedded systems must function despite limited storage and computational capacities in the same way as components of a wireless sensor network. A representative for Contiki usage is the ScatterWeb project.

The most important advantage of Contiki in comparison to TinyOS is the IP-communication (IPv6, IPv4), which is supported from the beginning. In Contiki the storage allocation happens during compiling. In contrast, the storage allocation for TinyOS is specified by the programmer, which means it is hard coded. A Contiki system consists of the kernel, the libraries, the program loader, and a set of processes, which can be an application or a service. The difference between an application and a service lies in the flexibility of the adaptations. In general, a service is programmed in a highly flexible manner so that different programs or applications can use it. In contrast the application is only developed for one special goal, e.g. health or building monitoring. In order to be flexible, Contiki supports different hardware such as Tmote Sky, JCreate, TelosB,MicaZ, Scatterweb plattforms MSB and ESB [9]. The programming language is Java. The process and the drivers can be replaced without interrupting a running system. If the processes need to communicate with each other they go through the kernel and communicate directly with the required hardware. The kernel itself is only responsible for event handling and outsources the remaining tasks to libraries that are linked if needed. In comparison to TinyOS the program code of Contiki consolidates all relevant packet imports and the application code itself in one file.

TinyOS code porting Contiki

In research TinyOS is the operating system of choice. The big advantages are the modular structure, the ongoing development, and the established community. A disadvantage is the fact that TinyOS requires special hardware such as Berkeley Motes that narrows the application field. But if an implementation runs under TinyOS with little complexity, the code and hardware can be transferred to Contiki. In order to realize the code porting, developed modules under TinyOS need to be subdivided into application and service processes in Contiki. The underlying wiring of components in TinyOS needs to be replaced by services interfaces in Contiki. If the code of TinyOS is adapted to the code requirements of Contiki, everything runs under the other operating system. For example, the TelosB nodes represent a platform, which can be programmed with TinyOS and Contiki, if the required drivers are supported.

IMPLEMENTATION

As shown in Section 4 the application area of wireless sensor networks is manifold but the main tasks - collecting data and transporting it to a sink - are the same; the intermediate operations and protocols can vary. A home monitoring scenario was chosen to validate the TinyIPFIX protocol and its extensions. The implemented protocols - TinyIPFIX and its extentions - are flexible in order to support different applications and hardware vendors at the same time, as mentioned in Section 4. They currently require the operating system TinyOS 2.x. Protocols and functionalities (e.g. UDP-Shell, certificate creation) can be ported to another operating system (e.g. Contiki) with little changes as well as used with other hardware. Throughout the wireless sensor network experiments in this dissertation the operating system TinyOS 2.1.1 with BLIP support is used as the operating system of choice together with Berkeley Motes IRIS and TelosB, which support IEEE 802.15.4/ZigBee and work on the 2.4 GHz band. For the IRIS platform two different sensor boards are available. The MTS400 has either a temperature and humidity sensor combined or a barometric pressure combined with a temperature sensor on board as well as a light sensor and voltage. GPS is optional and included in MTS420. The MTS300 includes sensors for light, temperature, and acoustic together with an acoustic actuator. Due to the physical positioning of the sensors on the board MTS300, it is not allowed to activate the temperature and light sensor at the same time, because the activation can damage both sensors. If an application uses the MTS300, either the 3-tuple temperature sensor, acoustic sensor and acoustic actuator are activated or the 3-tuple light sensor, acoustic sensor and acoustic actuator. This fact must be taken into account when deploying the wireless sensor network. For technical details it is referred to the MTS/MDA Sensor Board Users Manual from Crossbow Inc.

The TelosB node produced by the company Advantic is a MTM-CM5000-MSP sensor node with an external antenna including on board sensors for temperature, voltage, and humidity. The TelosB platform is also offered by Crossbow Inc., but not used here to prove the protocol's support in a wireless sensor network consisting of different vendors' hardware. In addition, TelosB nodes produced by Advantic include an antenna with a better radio range compared to TelosB node producedby Crossbow Inc., which doubles the radio range to 300 m outdoors and 40-50 m indoors.

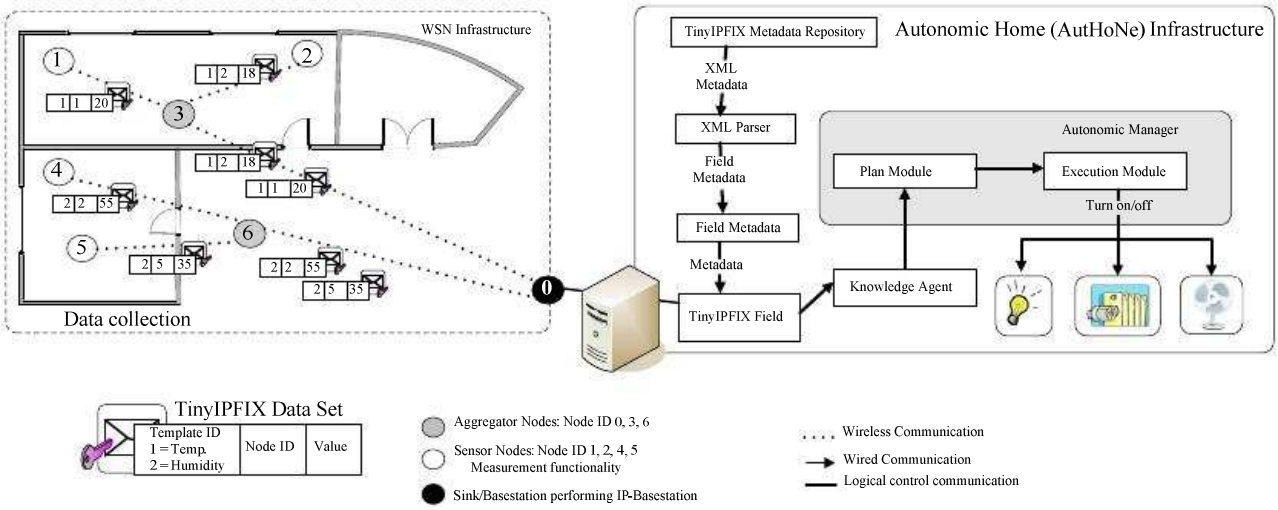

Figure 2. AutHoNe setup: simplified TinyIPFIX message structure

The assumed basic scenario for the implementation description is illustrated in Figure 2. The left part of the figure shows a building scenario, as assumed in the AutHoNe project, consisting of three rooms where sensor nodes are deployed. Here sensor nodes (marked white) are only data collectors and aggregator nodes (marked grey) forward received individual messages without modification towards the sink (node ID 0). The functionality of intermediary nodes (marked grey) can differ depending on their performed protocols (e.g. message/data aggregation). After collected data is received at the sink, the data is forwarded to the server. The server includes the required infrastructure in order to translate the received data by using XML-based meta data. The translated data is further forwarded to applications, such as the Knowledge Agents as part of the AutHoNe infrastructure, in order to make the collected sensor data available for management units (e.g. Autonomic Manager to coordinate functionality of lightning or heating control). The above described wireless sensor network consists of various hardware from different vendors in order to proof the flexibility of the developed protocols independent of the application scenario in this dissertation. Therefore, protocols used must allow integration of new vendors or setups (e.g. other sensor tuples) with a minimum of manual configuration. The established wireless sensor network uses IPv6 for communication purposes, because it was decided that BLIP is the IP communication support of choice for the experiments in this dissertation, which is included in the operation system TinyOS 2.1.1 as an existing

implementation. Today, in the Internet of Things it is assumed that networks use IPv6 instead of IPv4, where different arguments exist for this decision such as extended address space. If IPv4 is required on the server side, a parser must be integrated on the server for translation purposes. In the presented wireless sensor network experiments UDP is chosen as the transport protocol of choice, because todays' available wireless sensor network deployments prefer to use UDP.

Data transmission could include several Data Records, because the maximum pay load size of 102 bytes is not scooped. Assuming the additional Data Records refer to the equal Template, three additional Data Records would fit into the message. With the underlying BLIP stack the maximum IPFIX payload size can be expanded up to 1,024 bytes, because BLIP supports packet fragmentation. Recorded payload of the template transmission (marked red dashed) shown in Figure 3 is 39 bytes long and can be decoded as follows:

{04 27 16} ---> TinyIPFIX header in aggressive compression format whereas {27} indicates a total TinyIPFIX payload of size 39 bytes.
{01 00} ---> Set ID (here: 256)
{00 04} ---> Number of Template Fields (here: 4)
{80 a0 00 02 f0 aa 00 aa}--->Template Fields for Temperature value which include the following information:
{80 a0} ---> Type ID
{00 02} ---> Data Length ID in bytes
{f0 aa 00 aa} ---> Enterprise ID (here: 403767130)
{80 a1 00 02 f0 aa 00 aa} ---> Template Fields for Sound value
{80 a4 00 04 f0 aa 00 aa} ---> Template Fields for Node Time
{80 a5 00 02 f0 aa 00 aa} ---> Template Fields for Node ID

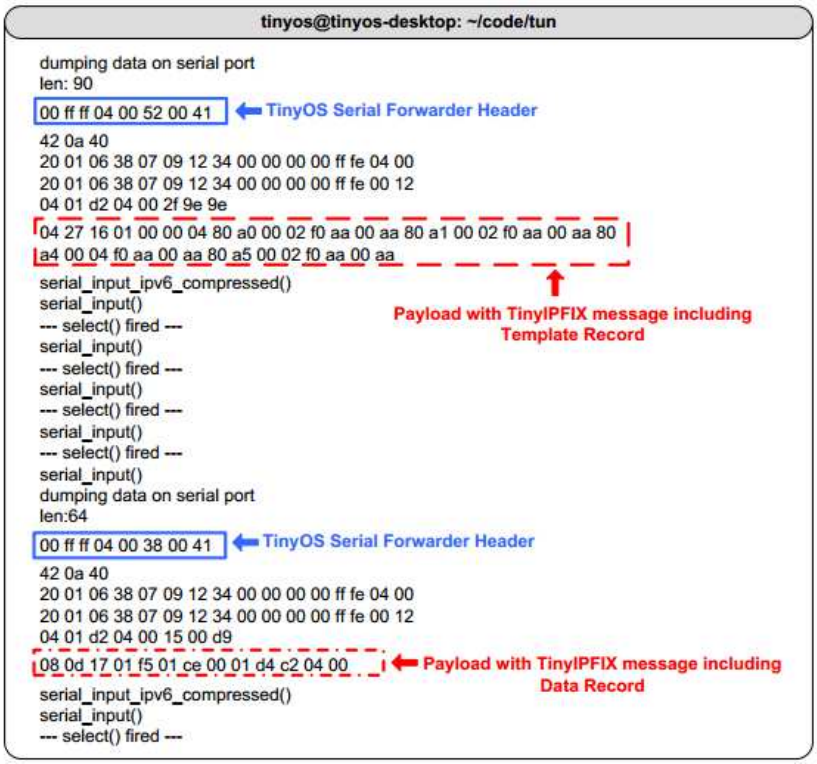

Figure 3. Wireless sensor network special case

CONCLUSION

In this paper, sensor data (e.g. temperature, brightness, acoustic, humidity) is sent to a gateway, which offers different possibilities for analysis in order to manage the environmental conditions of the building based on the habitants' preferences. The analysis' result can either be directly applied to different entities controlling automatic systems such as those developed for the AutHoNe project. Another possibility is the export of the data to analysis tools, which allow an import of data into a visualization tool in order to display the current status and to help optimizing the carbon footprint due to real-time feedback for the habitant's influencing their behavior.

REFERENCES

[1] P. Levis; N. Lee; M. Welsh; D. Culler, in Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, **2003**, 126-137.
[2] S. Fouladgar; B. Mainaud; K. Masmoudi; H. Afifi, in Proceedings of the 3rd European conference on Security and Privacy in Ad-Hoc and Sensor Networks, **2006**, 32-42.
[3] T. Kothmayr; C. Schmitt; W. Hu; M. Brünig; G. Carle, in 37th IEEE International Workshop on Practical Issues in Building Sensor Network Applications, **2012**, 69-86.
[4] Alan Demers; Johannes Gehrke; Biswanath P; Proceedings of the Conference. Innovative Data Systems Research(CIDR), **2007**, 412–422.
[5] J. Hui; D. Culler, *Proceedings of the IEEE*, **2010**(98), 1865-1878.
[6] Nihal Dindar; Peter M. Fischer; Merve Soner; Nesime Tatbul; Proceedings of the 5th ACM international conference on Distributed event-based system, **2011**(5), 33-69.
[7] Erik Eiesland; Master Thesis, Department of Computer Science, STFOLD University College, **2011**, 1-124.
[8] Marcelo R. N. Mendes; Pedro Bizarro; Paulo Marques; Proceedings of the second international conference on Distributed event-based systems, **2008**, 313–316.
[9] Trimurti Lambat; Sujata Deo; Tomleshkumar Deshmukh; *Journal of Chemical and Pharmaceutical Research*, **2014**(4), 888-892.