TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

IE70LT
Dmitri Ivanov 131120IVEM

SYNCHRONIZATION OF MEASUREMENT AND DATA TRANSFER IN 6LOWPAN NETWORK

Master's Thesis

Supervisor: Eero Haldre
Dipl.- Ing.
Senior Engineer

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

IE70LT
Dmitri Ivanov 131120IVEM

MÕÕTMISTE SÜNKRONISEERIMINE JA ANDMEEDASTUS 6LOWPAN VÕRGUS

Magistritöö

Juhendaja: Eero Haldre
Dipl. ins.
Vaneminsener

Tallinn 2017

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Dmitri Ivanov

07.05.2017

Abstract

The interconnection of wireless sensor networks with Internet could give a big benefit in data collection with further processing and ease to control and manage end devices. With the Internet protocol version 6 each end device connected to IPv6 network could have its worldwide unique address. To solve a problems how to interconnect wireless sensor networks with IPv6 and to define standards the Internet Engineering Task Force (IETF) established a special task group - IPv6 over Low power Wireless Personal Area Networks (6LoWPAN). In distributed wireless sensor networks synchronizing of the end devices is very attractive feature and critical in many industrial, medical and environmental applications. However there is no available good solution how to synchronize nodes in 6LoWPAN network. The result of this thesis is to propose synchronization solution using core concepts of PTPv2, test 6LoWPAN data transfer capabilities, test and measure implemented synchronization solution.

This MSc thesis reviews the main details of 6LoWPAN, describes hardware and software used to test realized synchronization process and data transfer capabilities. Implemented synchronization solution is described and measurements results are presented. The possible improvements are described in concluding section of this MSc thesis.

This thesis is written in English and is 71 pages long, including 6 chapters, 20 figures and 10 tables.

Annotatsioon

Mõõtmiste sünkroniseerimine ja andmeedastus 6LoWPAN võrgus

Traadita andurite võrgu ühendus Internetiga annab suurt kasu andmehõives koos hilisema töötlemisega, võimaldab lihtsustada lõppseadmete juhtimist ja haldamist. Interneti protokolli versioon 6 võimaldab anda unikaalse aadressi igale IPv6 võrguga ühendatud seadmele. Andurite sidumiseks traadita IPv6 võrguga on loodoud spetsiaalne standard - IPv6 over Low power Wireless Personal Area Networks (6LoWPAN). Traadita andurite võrgus lõppseadmete sünkroniseerimine on väga oluline funktsioon ja kriitiline paljude tööstus-, meditsiini- ja keskkonna süsteemides. Siiski ei ole saadaval head lahendust, kuidas 6LoWPAN võrgus lõppseadmeid sünkroniseerida. Selle lõputöö tulemusena on loodud töötav lahendus lõpseadmete sünkroniseerimiseks kasutades PTPv2 põhist ideed, testitud on 6LoWPAN andmeedastuse võimalusi, sünkrooniseerimese ja mõõtmiste resultaate.

Antud magistritöö annab ülevaate 6LoWPAN detailidest, kirjeldab kasutatavat riist- ja tarkvara et testida valmistatud sünkroonimise protsessi ja andmeedastust. Looduud sünkroniseerimise lahendus on dokumenteeritud ja mõõtmiste tulemused esitatud. Võimalikud parandused on kirjeldatud käesoleva magistritöö kokkuvõttes.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 71 leheküljel, 6 peatükki, 20 joonist, 10 tabelit.

List of abbreviations and terms

TTU	Tallinn University of Technology
6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
IPv6	Internet Protocol version 6
RPL	Routing Protocol for Low-Power and Lossy Networks
UDP	User Datagram Protocol
DAG	Directed acyclic graph
DODAG	Direction-Oriented Directed Acyclic Graph
USB	Universal Serial Bus
IEEE	Institute of Electrical and Electronics Engineers
GPIO	General-purpose input/output
DIO	Digital input/output
GPT	General-purpose timer
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
MCU	Microcontroller Unit
SOC	System on chip
ACK	Acknowledge
PAN	Personal Area Network
AES	Advanced Encryption Standard
TCP	Transmission Control Protocol
COAP	Constrained Application Protocol
CIDR	Classless Inter-Domain Routing
IID	Interface identifier
SLAAC	Stateless address auto configuration
NDP	Neighbor Discovery Protocol
MAC	Media access control
PHY	Physical layer
MTU	Maximum transmission unit
DAD	Duplicate address detection
IOT	Internet of things
PTP	Precision time protocol

Table of contents

1 Introduction 11
1.1 Motivation 12
1.2 Thesis tasks 13
2 6LoWPAN in details 14
2.1 Internet protocol version 6 15
2.1.1 IPv6 header 15
2.1.2 IPv6 addresses 16
2.1.3 Auto configuration in IPv6 17
2.2 IEEE 802.15.4 18
2.2.1 IEEE 802.15.4 PHY 19
2.2.2 IEEE 802.15.4 MAC 20
2.3 6LoWPAN as adaptation layer in protocol stack 22
2.3.1 Routing in 6LoWPAN 23
2.3.2 Header compression 25
2.3.3 Fragmentation and reassembly 27
2.4 6LoWPAN vs ZigBee 27
2.5 Summery 28
3 Available equipment 29
3.1 Hardware choice 29
3.2 Software choice 33
3.2.1 6LoWPAN border router 35
3.2.2 Contiki MAC layer 37
3.2.3 Routing in Contiki 38
3.2.4 UDP Client-Server realization 38
3.3 Summery 40
4 Synchronization of end devices 41
4.1 PTPv2 message classes 41

4.2 PTPv2 device types ... 43
4.3 PTPv2 simple master-slave clock hierarchy ... 44
4.4 Implementation of modified PTPv2 ... 45
4.5 Summery ... 52
5 Measurements results ... 53
5.1 Oscilloscope measurements ... 54
5.2 Synchronization precision measurements with external signal ... 59
5.3 Time to synchronize ... 61
5.4 6LoWPAN data transfer capability test ... 63
5.5 Summary ... 66
6 Conclusion and future improvements ... 68
6.1 Future improvements ... 68
References ... 70

List of figures

Figure 1. IPv6 header format [24]. 16

Figure 2. Routing methods. 17

Figure 3. IEEE 802.15.4 and IEEE 802.11n spectrum comparison [25].................... 20

Figure 4. 6LoWPAN as adaptation layer in protocol stack. 22

Figure 5. Ratio of uncompressed header data and payload. 23

Figure 6. Differences between mesh-under and route-over [26]. 24

Figure 7. Three scenarios of IPv6 header compression [26]. 26

Figure 8. Launchxl-cc2650 board. 31

Figure 9. Raspberry Pi 3 Model B. 32

Figure 10. Complete 6LoWPAN test setup. 33

Figure 11. Network block-diagram with 6LBR in router mode. 36

Figure 12. 6LBR webserver illustrating network tree with three nodes connected to 6lbr. 36

Figure 13. PTPv2 simple master-slave clock hierarchy. 45

Figure 14. PTP message exchange. 47

Figure 15. Modified PTP message exchange. 48

Figure 16. Master clock code flowchart. 49

Figure 17. Slave clock code flowchart. 51

Figure 18. Data transfer capability test results. 64

Figure 19. Two nodes fail to simultaneously send 22 data bytes every 100 milliseconds. 65

Figure 20. Two nodes are able to simultaneously send 19 data bytes every 100 milliseconds. 66

List of tables

Table 1. Results of synchronization protocol version 1, node 1.................................... 55
Table 2. Results of synchronization protocol version 1, node 2.................................... 56
Table 3. Results of synchronization protocol version 1, node 3.................................... 57
Table 4. Results of synchronization protocol version 2, node 1.................................... 57
Table 5. Results of synchronization protocol version 2, node 2.................................... 58
Table 6. Results of synchronization protocol version 2, node 3.................................... 58
Table 7. Summery tables combining all results separately for both protocols............... 59
Table 8. Measurement results taken with external signal, for version 1. 60
Table 9. Measurement results taken with external signal, for version 2. 60
Table 10. Time to synchronization, version 1 and version 2.. 62

1 Introduction

A Wireless Sensor Network (WSN) or wireless personal area network (PAN) is a self-configuring network consisting of sensor nodes communicating using radio signals. Sensor nodes are mainly small computers with reduced functionality and limited memory, low data rate, finite power source and one or more sensors. The main purpose of WSN is to collect data through its sensor nodes and send this data to other networks by means of gateway (or border router) for later processing. The following characteristics make WSNs attractive:

- Wireless connection – the sensor nodes could be placed in locations that are difficult to access. It reduces the cost of installation and maintaining. Nodes could communicate with each other in order to exchange and process data. Multi-hop network could be deployed which gives greater area coverage.

- Self-organization – nodes could be programmed to run neighbor discovery and arrange themselves into an ad-hoc network, this makes easy to deploy, expand and maintain network. The mesh network topology is robust to failures – if one node breaks down it will not influence the network.

- Low-power – WSNs can be deployed in locations with no available power source. Low power radios, sleep modes and radio duty cycling make possible for node long life using battery source or energy harvesting techniques.

A Wireless sensor networks become more useful and popular in such application areas as agriculture, medicine – remote patients monitoring, automotive, smart grid – enabling smart meters, home automation, environmental monitoring, tracking systems and different kinds of machine-to-machine communication. Usually such applications require lower power consumption for long battery life and low data rate.

Wireless sensor networks are one of building blocks of the Internet of things. IoT is a network of physical objects that can interact with each other to share information, these objects are smart cities, -buildings, -industry, -health, -transport, -energy etc. IoT objects

are enabled by means of different technologies like RFID, nanoelectronics, sensing, cloud computing, storage and of course WSN.

The interconnection of wireless sensor networks with Internet could give a big benefit in data collection with further processing and ease to control and manage end devices. With the Internet protocol version 6 each end device connected to IPv6 network could have its worldwide unique address. To solve a problems how to interconnect wireless sensor networks with IPv6 and to define standards the Internet Engineering Task Force (IETF) established a special task group - IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [1] [2] [3]. The new task group defined the fundamentals principles – header compression and encapsulation techniques which allow to send and receive IPv6 packets over IEEE 802.15.4 based networks.

1.1 Motivation

In distributed wireless sensor networks synchronizing of the nodes is very desirable property. In many industrial, medical and environmental applications accurate timestamping is critical for later data analysis or for making decision of control in real-time systems. Time synchronization is used in the design and operation of measurement, control, and communication systems. These systems share a common feature in that they interact with devices or processes that themselves operate based on real-world time [4]. Also time synchronization is very important in wireless sensor networks in which protocols and applications require precise time, for example, forming an energy-efficient radio schedule, conducting in-network processing (data fusion, data suppression, data reduction, etc.), distributing an acoustic beamforming array, performing acoustic ranging (i.e., measuring the time of flight of sound), logging causal events during system debugging, and querying a distributed database [5].

In spite of ubiquitous usage of time synchronization, there is hard to find any realization for 6LoWPAN networks. But I believe that with a growing popularity of IoT the interest to this topic among engineers and businesses will increase.

1.2 Thesis tasks

Before starting the work on thesis I defined main goals to be accomplished, additionally some goals were established in process. Below is the list of goals:

- In details study 6LoWPAN technology.
- Get familiar with synchronization protocols and choose to work with.
- Review available hardware and software solutions for 6LoWPAN and choose ones to work with.
- Deploy 6LoWPAN test environment.
- Implement synchronization protocol.
- Test and measurements of implemented synchronization solution and capabilities of 6LoWPAN based network.

2 6LoWPAN in details

Every end device connected to IPv6 network could have its own worldwide unique address. 6LoWPAN emerged to solve a problems of interconnecting wireless sensor networks with IPv6. 6LoWPAN is a networking technology or better to say adaptation layer that allows to efficiently transmit IPv6 packets within small link layer frames, such as defined by IEEE 802.15.4 standard.

The main characteristics of 6LoWPAN are following:

- Low bandwidth - data rates of 250 kbps for physical layer employing 2.4GHz band.
- Mesh or star network topology.
- Small packet size - the maximum physical layer frame is 127 bytes.
- Several addressing modes - either IEEE 64-bit extended addresses or (after an association event) 16-bit addresses unique within the PAN.
- Large number of devices.
- Unreliable devices due to variety of reasons: uncertain radio connectivity, battery drain, device lockups, physical tampering, etc.
- Sleeping mode – devices may use sleeping mode to save energy, however they are unreliable to communicate during sleep periods.
- Security – IEEE 802.15.4 describe only link-layer AES based security.
- Open standards including TCP, UDP, HTTP, COAP, MQTT, and websockets.
- End-to-end IP addressable nodes.

In this chapter the main parts of 6LoWPAN will be explained more detailed. I will describe the basics of Internet protocol version 6, 6LoWPAN network architecture, topology and protocol stack, also IEEE 802.15.4 will be reviewed.

2.1 Internet protocol version 6

Internet protocol version 6 or simply IPv6 [6] is a last version of Internet protocol, it is not compatible with well-known Internet protocol version 4 (IPv4). IPv6 uses addresses of length 128 bits, when IPv4 address is 32 bit long, however version 4 is still main Internet protocol and will remain for long time. According to ISO/OSI [7] network model IPv6 is located in layer 3 – network layer. The data handled by this layer is called packet. Generally IPv6 packet consists of control information needed to deliver data for recipient and of course useful data that should be transmitted.

2.1.1 IPv6 header

For better understanding of how 6LoWPAN uses IPv6 some details regarding IPv6 header should be studied. The header has a fixed size of 40 octets (320 bits) followed by upper layer (transport layer, however it could be lower layer data, like ICMPv6) data and optionally extension headers might be used. Comparing to IPv4 header, there are several fields in IPv6 header that give some improvements:

- The number of fields is reduced from 12 to 8
- The fixed header size of 40 bytes is aligned with 64 bits, this allows routers to faster forward hardware-based packets
- Addresses length increased from 32 bits to 128 bits

In the Figure 1 the IPv6 header fields are shown. The fields in header are:

1. Version – in our case version 6
2. Traffic class – affect the treatment of packets in routers, for example priority
3. Flow label – affect the treatment of packets in routers
4. Payload length – indicates the length of subsequent data
5. Next header – indicated the subsequent protocol, for example TCP or UDP
6. Hop limit – defines maximum number of hops that header should pass, going from one network node to the next
7. 8. Source and destination addresses – 128 bit long address of packet source and destination

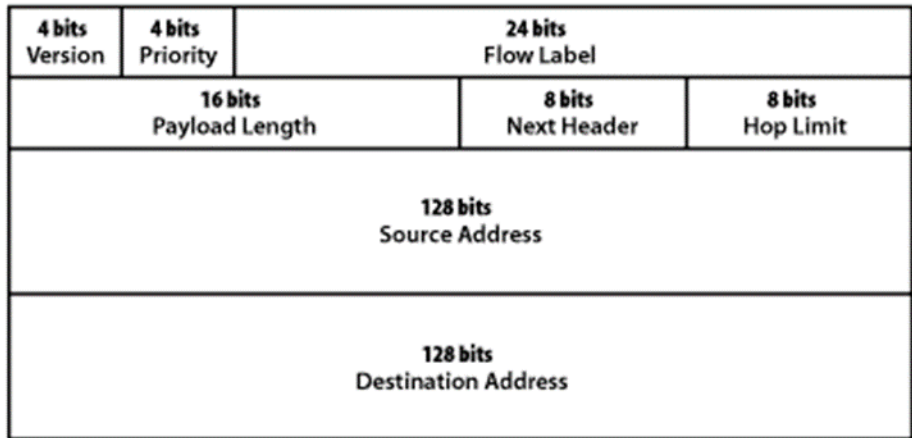

Figure 1. IPv6 header format [24].

The extension headers are always started with the basic IPv6 header following with chain of extension headers. The field next header is used to indicate following extension header. These headers give flexibility, as example, enable security by ciphering data in packet, enable authentication, support of communication using fragmented packets, mobility extension header is used to support mobile IPv6 service.

2.1.2 IPv6 addresses

An IPv6 addresses according to Classless Inter-Domain Routing (CIDR) procedure are divided into a host and network part. An IPv6 addresses are using hexadecimal notation, divided into blocks of 16 bit each and separated by colon, in total 8 groups; foremost zeros can be eliminated; blocks of successive zeros can be replaced by '::', this can be done only once.

An IPv6 network prefix specify some fixed bits and some non-fixed bits which can be used to create new sub-prefixes or to determine full IPv6 addresses assigned to the hosts. An IPv6 address is always linked to its interface, but never to a system (for example a PC). A host's interface in the local area network is identified by an interface identifier (IID), which is determined by the last 64 bits of IPv6 address.

An IPv6 addresses could be divided into four categories: unicast, anycast, multicast and reserved. Reserved addresses are maintained by IANA and listed in IANA IPv6 Special-Purpose Address Registry [8].

Below on the Figure 2 is shown graphical representation of routing methods: unicast, anycast and multicast.

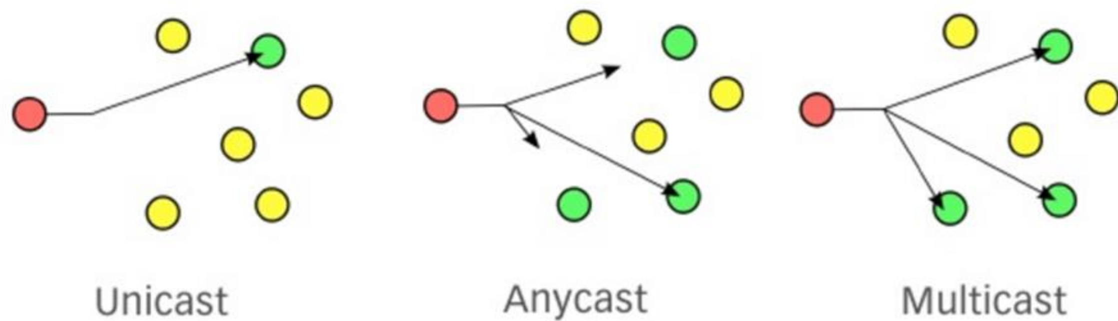

Figure 2. Routing methods.

Multicast addresses always start with prefix FF00::/8. Multicast addresses are only visible inside of its subnetwork, or with other configuration – worldwide. Multicast packets are sent from single host to multiple receivers.

Anycast addresses are used to transmit data packet from one source to the nearest destination from a group of interfaces. Anycast address does not have special prefix.

Unicast addresses always start with prefix FE80::/10. Unicast addresses could be sub-divided into Link-Local addresses and Global unicast addresses:

- Link-Local: only accessible in the own subnet, the router does not forward the packets to other networks. This type addresses consist of the prefix and the interface identifier (IDD).
- Global unicast: same as IPv4 public addresses, they are worldwide unique and they used to send a packet from one subnet to any destination in Internet.

2.1.3 Auto configuration in IPv6

An IPv6 addresses capacity is so huge that literally every device could have its own global unicast IPv6 address. One of advantages of Internet protocol version 6 is simplified address auto configuration mechanism. A new technique, which is not present in IPv4, is now allowed in IPv6 due to huge amount of available addresses is SLAAC – stateless address auto configuration. This mechanism is described in RFC4862 [9], it allows device to automatically generate an IPv6 address using a border router which gives connectivity to the network. With advantage of SLAAC wireless sensors could be added to network without any human intervention to configure IP devices, routers and servers.

Neighbour Discovery Protocol (NDP) [10] is used to auto configure addresses for end-devices. This protocol defines four messages:

- Router Advertisement
- Router Solicitation
- Neighbour Advertisement
- Neighbour Solicitation

The router is periodically sending Router Advertisement message containing configuration information to its end devices. If a node wants to join the network, it assigns itself a link-local address and sends out this address in Neighbour Solicitation message to all devices in given subnet to check if this address is not in use by other device. After that, node is waiting for some predefined time, if no Neighbour Advertisement message is received during given timeframe, node assumes that this address is unique. Now node sends Router Advertisement message to router and it replies with Router Advertisement message containing correct network prefix. Using this process any node in subnet can assign itself a worldwide unique IPv6 address.

2.2 IEEE 802.15.4

The 6LoWPAN allows to send and receive IPv6 packets over IEEE 802.15.4 based networks. The IEEE 802.15.4 [11] is a standard for low-rate wireless networks, it is maintained by IEEE 802.15 working group. This technical standard specifies the physical layer (PHY) and media access control (MAC) sublayer specifications for low-rate wireless connectivity with portable, fixed and moving devices with limited power consumption requirements. Physical layers are using license-free bands in a variety of geographic regions. The goal of IEEE 802.15.4 is to offer lower network layers of wireless personal area network to low cost, low power consumption, low complexity and low data rate wireless devices.

The IEEE 802.15.4 standard defines two types of devices that can participate in network. First one is full-function device (FFD) and second is reduce-function device (RFD). A full-function device is capable to serve as personal area network coordinator. A reduce-function device is not capable to serve as personal area network coordinator. An RFD is able to communicate only with FFD, while FFD can communicate with RFDs and other FFDs. An RFD is an end device while FFD is border router in wireless sensor network.

2.2.1 IEEE 802.15.4 PHY

The PHY layer provides interface between the MAC layer and the physical radio channel. The physical layer is responsible for the following tasks [11]:

- Activation and deactivation of the radio transceiver
- Energy detection (ED) within the current channel
- Link quality indicator (LQI) for received packets
- Clear channel assessment (CCA) for carrier sense multiple access with collision avoidance (CSMA-CA)
- Channel frequency selection
- Data transmission and reception
- Precision ranging for ultra-wide band (UWB) PHYs

The IEEE 802.15.4 standard specifies large number of physical layers based on different modulation methods and working in different operating bands. Frequency bands could be divided in groups of sub-GHz (operating band is under 1 GHz), high-band 2.4 GHz (worldwide unlicensed band) and ultra-wide band (UWB) with 3-10 GHz frequency band. The full list is available in IEEE 802.15.4 technical description [11]. The most used three unlicensed frequency bands are:

- 868.0–868.6 MHz: Europe, allows one communication channel
- 902–928 MHz: North America, up to thirty communication channels
- 2400–2483.5 MHz: worldwide use, up to sixteen channels

In my work I use PHY layer operating in 2.4 GHz frequency band. This layer type is based on direct sequence spread spectrum (DSSS) technique and uses offset quadrature phase shift keying (O-QPSK) modulation. DSSS is a technique for data redundancy where one data bit is converted into a sequence of X bits (technique 'chipping'). The DSSS makes the signal resistant to noise and can also reconstruct the original data from a low signal through redundancy. The raw bit rate is 250 kbps. The frequency band is 2400–2483.5 (MHz) with 16 channels, the center frequency of these channels is defined as follows: $F_c = 2405 + 5 (k – 11)$ in megahertz, for $k = 11, 12, \dots 26$, where k is the channel number.

The IEEE 802.15.4 shares 2.4 GHz ISM band with IEEE 802.11n known as Wi-Fi, Bluetooth etc., so the interference might appear during communication. Channel spacing in IEEE 802.15.4 is 5 MHz, each channel is 2 MHz wide and there is no spectral overlap between channels. There are 4 IEEE 802.15.4 channels with numbers 15, 20, 25 and 26 which fall between the non-overlapping IEEE 802.11n channels. In my master work all measurements and tests were done using channel 25 and no tests regarding interference were done. According to the IEEE 802.11n standard there are three non-overlapping channels (channel 1, 6 and 11). In figure below is shown IEEE 802.15.4 and IEEE 802.11n spectrum comparison.

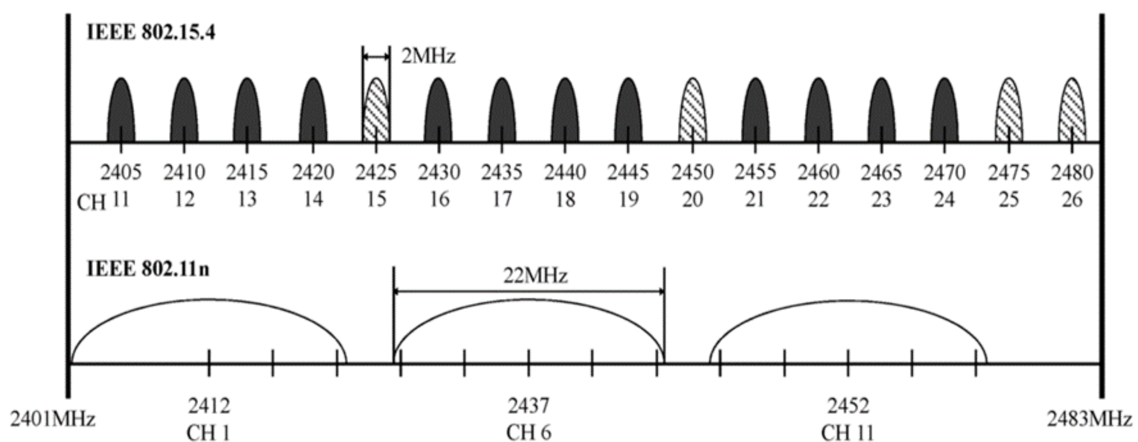

Figure 3. IEEE 802.15.4 and IEEE 802.11n spectrum comparison [25].

2.2.2 IEEE 802.15.4 MAC

The main role of MAC layer is to maintain and control the signal in a standardized manner to ensure network reliability. The medium access control layer provides an interface between the physical layer and the application layer, also it is responsible for the following tasks:

- Generating network beacons when device acting as network coordinator
- Synchronizing the beacons
- Supporting personal area network association and disassociation
- Providing device security
- Employing the CSMA-CA mechanism to access the channel
- Handling and maintaining the Guaranteed Time Slot (GTS)
- Providing a reliable link between two peer MAC entities

The medium access layer gives services for the application layer, two groups of services are used: the MAC Management Service - the MAC Layer Management Entity (MLME) and the MAC Data Service - the MAC Common Part Layer (MCPS).

The MLME provides the service interfaces through which layer management functions may be invoked. The MLME service is also responsible for maintaining a database of managed objects pertaining to the MAC layer. The MCPS allows the MLME to use the MAC data service.

The IEEE 802.15.4 standard defines four MAC frame types:

- Beacon frames
- Acknowledgment frames
- Data frames
- MAC control frames

Network coordinator uses beacon frames to describe channel access mechanism to other network participants. The data frame is used to send payload of different length (supported payload length is in range of 2 to 127 bytes). To increase reliability for data frame and control frame transmissions the acknowledgment frames are used. The MAC control frames are used to support network management functions, such as association and disassociation to/from the wireless network.

The medium access layer can operate in beacon-enabled and non-beacon modes. With the beacon-enabled mode the coordinator periodically transmit beacons to synchronize nodes that are associated with it and nodes in the network are watching for these beacons. In this mode all the devices in a network know when to communicate with each other. In a non-beacon mode coordinator does not send beacons. In this case when a node wants to transmit, first it will make sure that channel is available by using means of CSMA/CA. Once the channel is in idle state, node sends out data frame after some random back off time. Receiver replies with acknowledgment frame, if the sender does not receive the ACK frame it tries to transmit data once again until acknowledgement or until the maximum number of retransmissions reached.

2.3 6LoWPAN as adaptation layer in protocol stack

Between the IP (network layer) and the lower layers (MAC and PHY) some important problems need to be solved, these problems are solved by means of an adaptation layer, the 6LoWPAN. The 6LoWPAN protocol stack is similar to a standard IP stack, but with some differences. 6LoWPAN supports stateless compression of IPv6 (network layer) and higher layer headers such as UDP (transport layer) along with fragmentation and mesh addressing features. UDP is most used transport layer protocol for data flow in 6LoWPAN networks. 6LoWPAN only defines operation of IPv6 over the IEEE 802.15.4 standard, devices in 6LoWPAN network acting as border routers, may support IPv6 tunnelling mechanisms to connect 6LoWPAN networks with IPv4 networks. Below you can see the Figure 4 showing the 6LoWPAN as adaptation layer in complete protocol stack.

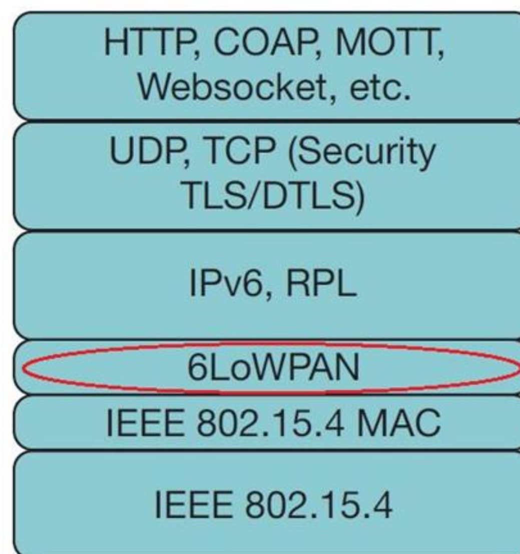

Figure 4. 6LoWPAN as adaptation layer in protocol stack.

The main goals of adaptation layer are:

- Fragmentation and reassembly - IPv6 specification defines the maximum transmission unit (MTU) of 1280 bytes. This is the minimum value the MAC layer has to provide to be able to send IPv6 packets without fragmentation. In IEEE 802.15.4 standard the protocol data units are much less in size. To solve such a difference a fragmentation and reassembly mechanisms are used at the layer below IP.
- Mesh routing protocol - support of a multi-hop mesh network.

- Address auto configuration - techniques to create IPv6 stateless address auto configuration. This technique generates the IPv6 Interface Identifier (IID) from the EUI-64 assigned to the IEEE 802.15.4 device. It is very attractive feature of 6LoWPAN because it reduces the configuration overhead on the end devices.
- Header compression – the maximum available payload according to the IEEE 802.15.4 specification is 127 bytes, from this value up to 25 bytes for the MAC header and 40 bytes for the IPv6 header have to be subtracted. This gives that in worst case only 62 bytes remain for payload, however if security or other protocols (such as UDP or TCP) are added – the space for useful data payload increased even more. Figure 5 shows the relation of uncompressed header data and actual payload.

127 Byte			
MAC Header	IPv6 Header	UDP Header	Payload
25 Byte	40 Byte	8 Byte	54 Byte

Figure 5. Ratio of uncompressed header data and payload.

2.3.1 Routing in 6LoWPAN

To send a data packet from one node to another, sometimes through multiple hops, the routing techniques are in use. Two different routing categories are defined: mesh-under and route over. Mesh-under uses the layer-two (link layer) addresses (IEEE 802.15.4 MAC or short address) and route-over uses layer three (network layer) addresses (IP addresses). The differences of mesh-under and route-over are illustrated in Figure 6 below.

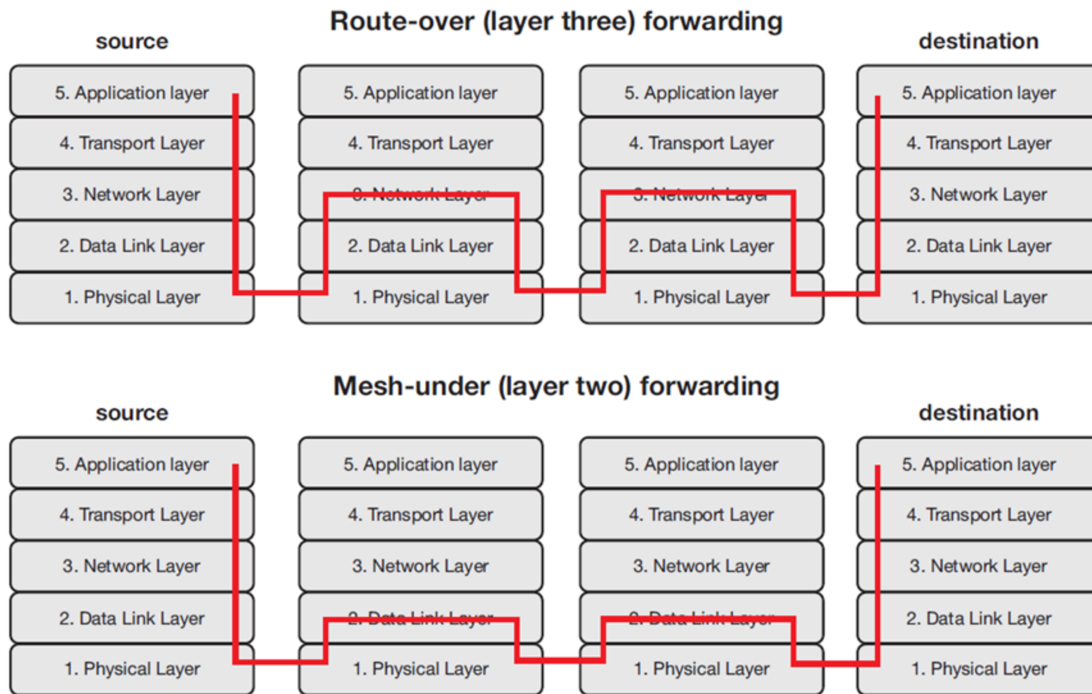

Figure 6. Differences between mesh-under and route-over [26].

In case of a mesh-under, data packets routing is done transparently, so these networks are considered to be one IP subnet. The only IP router in such a system is the edge router. In mesh-under network broadcast domain is used to guarantee compatibility with higher layer protocols, for example duplicate address detection (DAD) that making sure that no duplicate addresses exist. These messages are sent to every device in the network, which in turn highly increase network load. This makes mesh-under networks better suited for small and local networks.

In case of route-over networks routing is done in IP level, in this way each hop represents an IP router. The utilization of IP routing gives possibility to deploy more scalable and powerful networks, because each router must implement all functions supported by an ordinary IP router such as duplicate address detection. In route-over network messages are not broadcasted, but sent only to the nodes within the radio range. Protocol for route-over networks, known as RPL (IPv6 routing protocol for low-power and lossy networks), is the most used in 6LoWPAN networks. The advantage of route-over, compared to mesh-under, is that most of the protocols used on a standard TCP/IP stack is possible to implement and use as is.

RPL supports two different routing modes: storing mode and non-storing mode. When storing mode is used all devices in the network configured as routers hold routing and a neighbour table. The neighbour table is used to keep track of a node's direct neighbours

and the routing table is used to search routes to devices. In a non-storing mode only the edge router keeps routing table. This means that when data packet needs to be sent from one node to another inside the same network, first the packet is sent from source node to the edge router, which looks up for complete route to destination in its routing table and adds this destination to packet. In case of non-storing mode the network overhead increases with the number of hops that packet should travel to reach the destination, while using storing mode requires more recourses of every device in network acting as router.

RPL is a proactive distance vector protocol, when RPL network is initialized, it immediately starts to find the routes. RPL builds Destination Oriented DAGs (DODAGs) rooted in direction to one sink called DAG ROOT. DODAG is identified by a unique identifier DODAGID. Objective Function (OF) metric indicates the dynamic constraints and has different metrics like hop count, parent selection, expected transmission count, latency, energy consumption, etc. By using objective functions DODAGs could be optimized. Each node has its rank number, by this number it is possible to determine node's relative position and distance to the root in the DODAG. RPL specifies three control messages to exchange graph related information [12]: DODAG Information Solicitation (DIS), DODAG Information Object (DIO), and DODAG Destination Advertisement Object (DAO).

2.3.2 Header compression

The traditional way of header compression is status based, which is used at point-to-point connections where a flow between two end points is stable. This method is effective in static networks with stable links, but this method is not effective or not applicable at all to dynamically changing networks with multiple hops like one based on 6LoWPAN standard. In 6LoWPAN two different methods are used for header compression – IPv6 header compression (IPHC) and next header compression (NHP). IPv6 header compression relies on information referring to the entire 6LoWPAN. IPv6 header compression assumes the following will be the common case for 6LoWPAN network:

- Internet protocol version is 6.
- Traffic Class and Flow Label are both zero.
- Payload Length can be inferred from lower layers: from fragmentation header or from the IEEE 802.15.4 header.

- Hop Limit will be set to a known value by the sender.
- Addresses assigned to devices in network will be formed using the link-local prefix or a small set of routable prefixes assigned to the entire 6LoWPAN.
- Addresses assigned to devices are formed with an interface ID derived from either the 64-bit extended or from the IEEE 802.15.4 16-bit short addresses

Figure 7 below shows three possible scenarios of header compression. In the first example communication is carried between two devices in the same network, when link-local addresses are used, the header can be compressed to size of 2 bytes. In the second example communication is carried between devices of different networks and destination network prefix is known – in this case header is compressed up to 12 bytes. The third example is similar to second, but without knowing of destination prefix – in this case the header is compressed to size of 20 bytes. The first example can't be used for sending application data, as it can be used to send data only to direct neighbors, however this header compression scenario is very important for routing protocol. The third example is worst case, but it still gives a 50 percent header compression ratio. In these examples UDP header compression is not described, however it is part of 6LoWPAN standard. UDP header is possible to reduce from 8 bytes up to 1 byte.

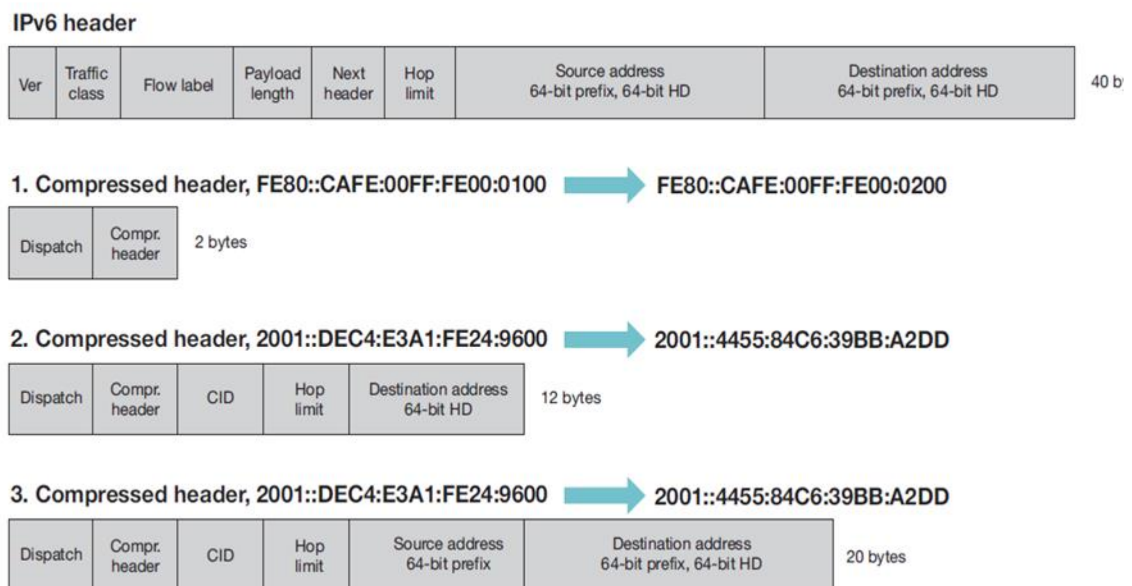

Figure 7. Three scenarios of IPv6 header compression [26].

2.3.3 Fragmentation and reassembly

One of the 6LoWPAN protocol suite components is fragmentation, which fulfils the IPv6 MTU requirement of 1280 bytes [6]. Fragmentation requires generating additional data in each packet, this is needed to correctly reassemble packet at destination. During reassemble the additional data is removed and packets are combined to a complete package.

The fragmentation process depends on which routing mode is used – mesh-under or route-over. In case of mesh-under routing, fragments are reassembled only at their final destination, but in case of route-over routing data packets are reassembled at every hop. In case of route-over every device acting as router should have enough memory space and recourses to store all fragments, since all fragments are assembled and the complete packet is analysed to determine the next destination node. In case of mesh-under network all packets are processed immediately, this mean that a lot of traffic is generated. This may lead to situation when fragments are missed during reassemble, in this case the complete packet should be retransmitted. In general, fragmentation should be avoided every time when it is possible, because it has negative influence on the battery life of a power-constrained device.

2.4 6LoWPAN vs ZigBee

6LoWPAN is not only one standard for deploying wireless sensor networks and it is not only one that uses the IEEE 802.15.4 standard. The most usually 6LoWPAN is compared to its competitor - ZigBee. ZigBee is also low data-rate, low-cost and low-power wireless mesh networking standard designed for power-constrained devices. It is typically implemented for personal or home-area networks, for smart energy, in commercial building automation, or in a wireless mesh for networks that operate over longer ranges. Unlike 6LoWPAN, ZigBee cannot easily communicate with IPv6.

To better understand main differences of both standards it is better to compare them according to ISO/OSI networking model. ZigBee is located on layers 3 to 6 (network - application), while 6LoWPAN only between layers 2 and 3 (data-link and network). Since ZigBee uses more network layers the complete system becomes more complex. ZigBee devices have limited interoperability, they can communicate only with other ZigBee devices and must have ZigBee Alliance license. 6LoWPAN offers interoperability with

any wireless 802.15.4 devices and moreover with devices on any other IP network link (e.g., Ethernet or Wi-Fi) with a simple bridge device.

2.5 Summery

In this chapter the main parts of 6LoWPAN were reviewed. Latest Internet protocol version 6 was described in sufficient way for understanding how it is used in 6LoWPAN, the main aspects of IPv6 such as header, address and auto configuration were reviewed. For IEEE 802.15.4 standard were covered physical layer responsibility areas and frequency bands; described MAC layer tasks, frame types and operating modes such as beacon-enabled and non-beacon. For 6LoWPAN adaptation layer routing, header compression, fragmentation and reassembly were also described.

In the next chapter I will introduce to the reader equipment I used, main software parts and tools, also some 6LoWPAN parts will be covered in more details.

3 Available equipment

In this chapter I will describe hardware, software and tools I used in this master work. Mostly I will pay attention on software – describe Contiki OS and some 6LoWPAN applications I tested. In this chapter I will not review precision synchronization protocols I developed during this work, this I leave for next chapter 4.

With a rapid grow of interest to IoT almost all main microcontroller and semiconductor manufacturers started to produce their own systems on chip (SoC) or separate radio chips supporting IEEE 802.15.4. To start working with or evaluate IEEE 802.15.4 it is better to have already complete evaluation board, so engineers do not need to design their own platforms, this always speed up development process and get faster familiar with technology, in my case 6LoWPAN based on IEEE 802.15.4 standard.

3.1 Hardware choice

For me the main aspects choosing the right platform were: the lowest price, easy to buy (availability), real examples of working solutions, open software, programming tools and it best case supporting community working with same hardware and software. Spending some time choosing the suitable platform finally I stopped my choice on Texas Instruments Launchxl-cc2650. This is well-known LaunchPad (previously I worked with another LaunchPad) with multi-standard wireless MCU on board. The attractiveness of this evaluation board is its low price, only 25 Euros (the cheapest among competitors), available low level driver libraries by TI, no external programming tool, 32 bit ARM processor and active TI's engineering community. For development and tests I bought 4 Launchxl-cc2650 boards.

The CC2650 is a SimpleLink multi-standard 2.4 GHz ultra-low power wireless MCU. The device is a member of the CC26xx family of cost-effective, ultralow power, 2.4-GHz RF devices. According to manufacturer it is very low-power providing operation on small coin cell batteries and in energy-harvesting applications. This MCU is called multi-standard because it supports Bluetooth, ZigBee and 6LoWPAN, and ZigBee RF4CE remote control applications. The Bluetooth Low Energy controller and the IEEE 802.15.4 MAC are embedded into ROM and are partly running on a separate ARM Cortex-M0 processor.

Here are some features of CC2650 derived from technical reference manual [13]:

- Powerful ARM® Cortex®-M3
- Up to 48-MHz Clock Speed
- 128KB of In-System Programmable Flash
- 20KB of Ultralow-Leakage SRAM
- Efficient Code Size Architecture, Placing Drivers, Bluetooth® Low Energy Controller, IEEE 802.15.4 MAC, and Bootloader in ROM
- Four General-Purpose Timer Modules (Eight 16-Bit or Four 32-Bit Timers, PWM Each)
- UART
- All Digital Peripheral Pins Can Be Routed to Any GPIO
- 12-Bit ADC, 200-ksamples/s, 8-Channel Analog MUX
- Real-Time Clock (RTC)
- AES-128 Security Module
- Normal Operation: 1.8 to 3.8 V
- Active-Mode RX: 5.9 mA
- Active-Mode TX at +5 dBm: 9.1 mA
- Excellent Receiver Sensitivity (–97 dBm for BLE and –100 dBm for 802.15.4), Selectivity, and Blocking Performance
- Programmable Output Power up to +5 dBm
- Various of supported Tools and Development Environment

Below is a Figure 8 where one launchxl-cc2650 board is depicted.

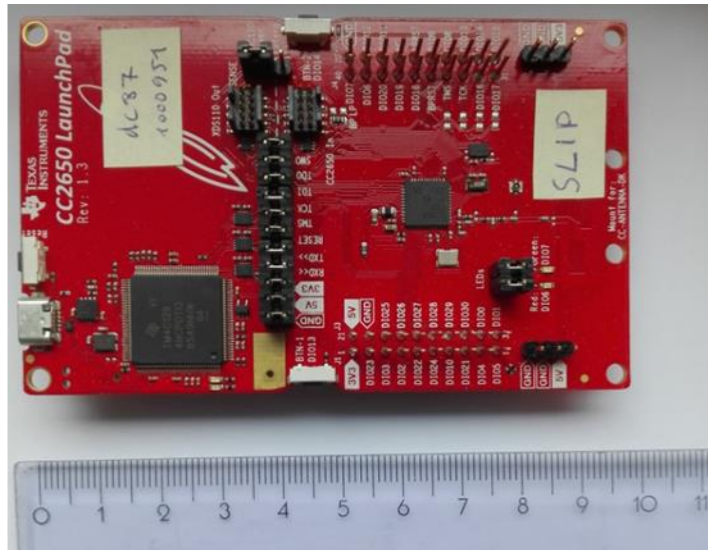

Figure 8. Launchxl-cc2650 board.

During development and tests the boards were used as end-devices, as boarder router, as serial line Internet Protocol (SLIP-radio) and as IEEE 802.15.4 packet sniffer. In case when it was used as slip-radio, it was connected to Raspberry Pi 3 which was acting as 6LoWPAN boarder router, the rest three launchxl-cc2650 boards were end-devices in network (nodes). When I tested synchronization protocols, one board was used as boarder router (server) and the rest three as end-devices (clients). When I used launchxl-cc2650 board as packet sniffer – it was connected to PC, one other launchxl-cc2650 board was acting as boarder router and the rest two was end-devices in 6LoWPAN network.

Specific firmware were used in every case. Flash programming were done with Texas Instruments' programming tool Flash Programmer 2 and XDS110 debugger (JTAG debugger for Texas Instruments' microcontrollers and embedded processors) already mounted on launchxl-cc2650 board.

To connect my 6LoWPAN network to the outside world the gateway with Ethernet interface is required. Here are also available different solutions, for example single Ethernet module with ENC28j60 which could communicate with launchxl-cc2650 board via I2C interface. But the better choice is to use more powerful single board computer like BeagleBone or Raspberry Pi. As I already had Raspberry Pi 3 then I decided to use it.

Here are some features of Raspberry Pi 3 model B:

- 1.2GHz 64-bit quad-core ARMv8 CPU

- 802.11n Wireless LAN
- Bluetooth Low Energy (BLE)
- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- Ethernet port
- Micro SD card slot

The Figure 9 shows the Raspberry Pi and reader could see its size.

Figure 9. Raspberry Pi 3 Model B.

Since Raspberry Pi is a single board computer, not a bare microcontroller, the operating system is required to simplify using Pi and eliminate low level programming. Several of operating systems could be used on Raspberry Pi, most of them are Linux based, however Microsoft also supports Raspberry Pi with Windows 10 IoT Core. But I choose to use Raspbian - official supported operating system.

On Raspbian OS I installed CETIC 6LBR (6LoWPAN boarder router) [14] – is a 6LoWPAN/RPL Border Router solution, or better to say it is a complete solution for interconnecting IP and 6LoWPAN networks. 6LBR expects an Ethernet interface on the IP side and an 802.15.4 radio interface on the 6LoWPAN side.

CETIC 6lbr requires external hardware supporting 802.15.4 radio. In my case this external hardware is launchxl-cc2650 board acting as slip-radio. On this launchxl-cc2650 executes a specific firmware implementing the communication protocol between the 6LBR process and the IEEE 802.15.4 radio. The data are transferred over UART. Some specifics of 6lbr, regarding 6LoWPAN, I will review in more details in the next chapter 3.2 describing software and firmware I used in this project. Below is Figure 10 showing complete 6LoWPAN test setup.

Figure 10. Complete 6LoWPAN test setup.

3.2 Software choice

As in case of choosing the right hardware, there are plenty of available software choices. My requirements to choose software are following. First I need to choose operating system that fits to CC2650. Secondly this operating system must have native support for a number of common IP-based protocols, this should significantly simplify application development and save time. Of course this operating system should be open sourced with full source code available. Among different embedded operating systems that I looked thru, the best that fits my needs is Contiki-OS [15]. It is the open source operating system for the Internet of Things. Contiki-OS is written in standard C, full source code is available, and also it has active community with contributors from different well-known

companies like Atmel, Cisco, ETH, Redwire LLC, SAP, Thingsquare, and many others. It has native support (drivers) for CC2650 MCU and moreover Texas Instruments supports Contiki OS [16].

The main Contiki-OS features are:

- Supports low-power IPv6 networking, including 6lowpan adaptation layer, RPL IPv6 multi-hop routing protocol and the CoAP RESTful application-layer protocol.
- Full IP Networking - provides standard IP protocols such as UDP, TCP, and HTTP.
- Designed for tiny and low-power systems.
- ContikiMAC radio duty cycling mechanism allows devices to sleep between packet transmissions.

Contiki uses a mechanism called protothreads [17]. Protothreads provide sequential flow of control without complex state machines or full multi-threading. With protothreads, event-handlers can be made to block, waiting for events to occur. Protothreads API consists of main operations like:

- PT_INIT () – initialize structure with protothreads description. Protothread should be initialized before execution starts.
- PT_BEGIN () – defines the begin of protothread within function
- PT_END () – defines the end of protothread within function
- PT_WAIT_UNTIL () and PT_WAIT_WHILE () – blocks the thread until some condition will come true
- PT_WAIT_THREAD () and PT_SPAWN () – awaits until subsidiaries protothreads finalize execution
- PT_RESTART () and PT_EXIT () – restart and exit protothread
- PT_SCHEDULE () – schedules (in fact launching) a protothread

- PT_YIELD () – yield from the current protothread, returns control to scheduler

3.2.1 6LoWPAN border router

Border router is required to connect 6LoWPAN network to traditional IP networks like Internet. As I already mentioned in chapter describing hardware I used in this work, the 6lbr was chosen for boarder router. Project 6lbr was developed by CETIC [14] - applied research center in the field of ICT located in Belgium, 6lbr is a Contiki-OS branch. 6LBR is based on uIP (micro IP) which optimized for embedded platforms.

Some features of 6lbr that makes it attractive:

- Synchronizes 6LoWPAN WSNs with IP network
- Network auto configuration
- Different network architectures
- NAT64 to provide bidirectional IPv4 connectivity
- An enhanced webserver with configuration commands

6LBR can work as stand-alone router on embedded hardware or on a Linux host. 6LBR is designed for flexibility, it can be configured to support various network topologies while smartly interconnecting the WSNs with the IP world [14].

6LBR can be used in three different modes: bridge, router and transparent bridge. In my work I used router mode. In this mode 6lbr acts as independent router interconnecting 2 IPv6 subnets. In this mode wireless sensor network is managed by the RPL protocol and Ethernet subnet is managed by IPv6 NDP. This operating mode could be understand as a gateway between Ethernet side and 6LoWPAN RPL. In the Figure 11 below depicted block diagram of network with 6lbr in router mode.

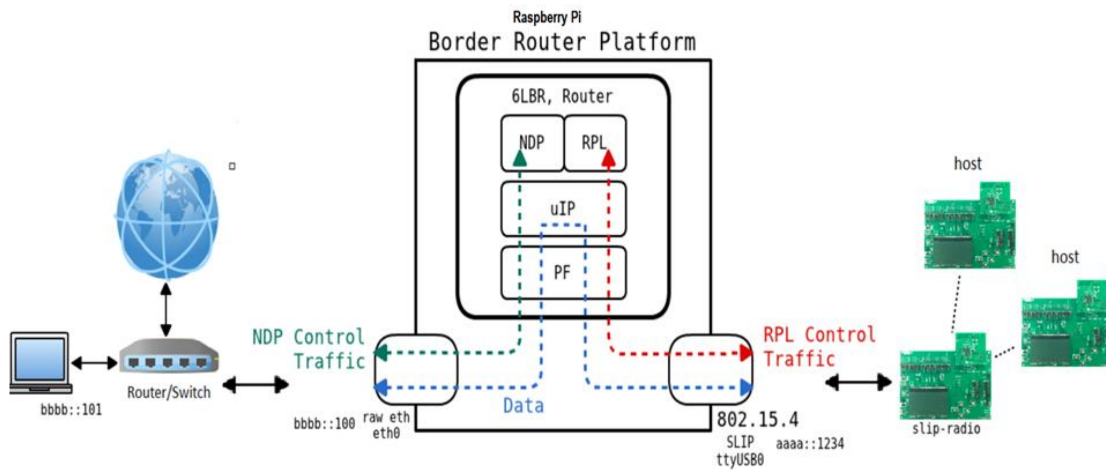

Figure 11. Network block-diagram with 6LBR in router mode.

6LBR has a built-in webserver which could be accessed from web browser, the default address is http://[bbbb::100]. Thru the webserver different network configurations could be done: configure IEEE 802.15.4 channel, configure security layer, configure global IP address to access from Internet, configure RPL behaviour etc. Also from webserver different statistics could be found, like connected sensors list or display node tree, up-time, amount of transmitted packets etc. Below in Figure 12 is shown webserver interface and example of node tree, with three 6LoWPAN nodes connected to the border router.

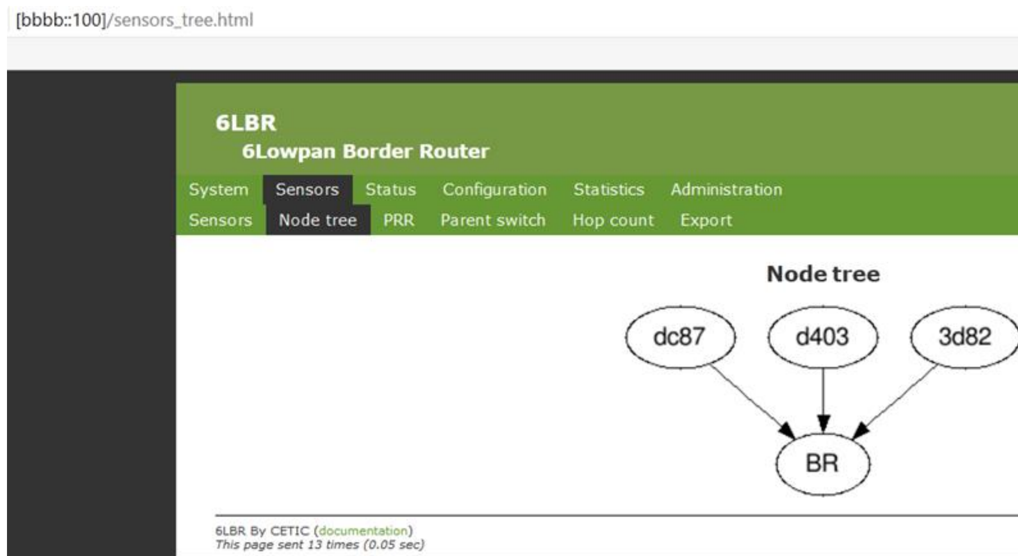

Figure 12. 6LBR webserver illustrating network tree with three nodes connected to 6lbr.

The firmware for launchxl-cc2650 acting as slip-radio was taken from Contiki's examples for slip-radio. Some minor modifications in configuration file were done to match the target board launchxl-cc2650. 6LBR code was used as is without modifications, only file static configuration file located in Raspbian */etc/6lbr/6lbr.conf* was modified: configured

router mode, selected the mode of the Ethernet-side interface and set the right baud rate to be used to communicate with the slip-radio.

3.2.2 Contiki MAC layer

The medium access in Contiki includes three different layers: medium access control (MAC), radio duty-cycle (RDC) and framer. Global variables *NETSTACK_FRAMER*, *NETSTACK_RDC and NETSTACK_MAC* in *core/net/netstack.h* can be used to access network layer for defining custom configuration.

Two MAC drivers are supported in Contiki, these are CSMA and NullMAC. CSMA is responsible for receiving incoming packets from radio duty-cycle layer and in same time uses radio duty-cycle to transmit packets. When the radio duty-cycle layer or radio layer find that the medium is busy, the MAC layer will retransmit packet after some backoff time. The medium access check is done by the radio duty-cycle driver. NullMAC is a simple pass-through protocol. It calls the appropriate radio duty-cycle functions. The CSMA in Contiki keeps a list of sequence numbers, retransmission, etc., while NullMAC doesn't do any of this and gives a less reliable network. In my work I used CSMA.

Contiki RDC driver manages the sleep period of nodes. This driver determine when packets are going to be transmitted and makes sure that nodes are awake when packets should be received. In Contiki following RDC drivers are supported ContikiMAC, X-MAC, LPP, NullRDC and SicslowMAC. Radio duty-cycle drivers try to keep the radio off as much as possible and only periodically waking up to check the wireless medium for radio activity. When RDC detects some activity, it turns on the radio to check if the packet should be received, other way it turns radio back in sleep mode. The check periodicity is set in Hz and is defined by global variable *NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE* located in *core/net/netstack.h*. NullRDC is a simple pass-through driver, it will never switch radio into sleep mode. Usage of NullRDC will give a lower latency, but higher receive energy consumption, whereas ContikiMAC has lower receive energy consumption, but higher latency and increased transmission energy. In my work I used NullRDC driver. The ContikiMAC radio duty cycling protocol is in details described by its developer in [18] paper.

The framer driver is a set of functions, which are responsible for constructing and parsing MAC header. The two most used framers are framer-802154 and framer-nullmac. In case

if framer-802154 is used the driver frames the data in compliance to the IEEE 802.15.4 standard. The framer-nullmac only fills in the two fields of nullmac_hdr: receiver address and sender address. In my work I used framer-802154.

3.2.3 Routing in Contiki

The default routing protocol in Contiki is RPL, its implementation could be found in *core/net/rpl*. Contiki automatically forms a wireless IPv6 network. The basics of RPL I already described in chapter 2.3.1 Routing in 6LoWPAN.

To enable routing in Contiki the following variable *UIP_CONF_ROUTER* should be enabled and to enable RPL (there are some other routing protocols supported by Contiki, like AODV) the variable *UIP_CONF_IPV6_RPL* is used. In my configuration I used RPL, disabled sending of routing advertisements, disabled IP forwarding and disabled RPL configuration statistics. By default Contiki uses storing mode for RPL downward routes, meaning that all nodes store in theirs routing tables the addresses of child nodes.

3.2.4 UDP Client-Server realization

In this chapter I would like to describe UDP client-server realization I implemented. Based on this code for server and for client I developed synchronization protocols and measurements tests were done. In this realization UDP server is acting as DAG Root and all clients have rank number one, meaning they directly connected to the root. So in this realization I didn't use Raspberry Pi with 6lbr on board, of course it is possible to use Raspberry Pi with 6lbr as UDP server, but I found it easier to use launchxl-cc2650 as server (in this case I don't need to reinstall 6lbr every time I make code changes).

First I need to choose global addresses for server and clients – I choose to acquire address from the IEEE 802.15.4 16-bit short addresses. This is done by calling function to construct address with right arguments: *uip_ip6addr(&ipaddr, UIP_DS6_DEFAULT_PREFIX, 0, 0, 0, 0, 0x00ff, 0xfe00, 1);* Constructed address is saved in *ipaddr*. Now on the server side I need to configure it as DAG root, this is done with the following code:

```
  struct uip_ds6_addr *root;
  uip_ds6_addr_add(&ipaddr, 0, ADDR_MANUAL);
  root = uip_ds6_addr_lookup(&ipaddr);
  if(root != NULL) {
    rpl_dag_t *dag;
    dag = rpl_set_root(RPL_DEFAULT_INSTANCE,(uip_ip6addr_t *)&ipaddr);
    uip_ip6addr(&ipaddr, UIP_DS6_DEFAULT_PREFIX, 0, 0, 0, 0, 0, 0, 0);
    rpl_set_prefix(dag, &ipaddr, 64);
  }    PRINTF("created a new RPL dag\n");
  } else {
    PRINTF("failed to create a new RPL DAG\n");
  }
```

Now for both, server and client I need to create new UDP connection and bind UDP ports. Binding is done for port which server/client is going to listen to, so for client I need to bind server's port and vice versa for server. This is accomplished by means of following functions (example for client):

```
static struct uip_udp_conn *server_conn_sync
server_conn_sync = udp_new(NULL, UIP_HTONS(UDP_CLIENT_PORT_SYNC), NULL);
udp_bind(server_conn_sync, UIP_HTONS(UDP_SERVER_PORT_SYNC));
```

Now server and client in main thread's while loop will check for function *uip_newdata()* if new packet is received, when new data is received I call *udp_handler()* function to make some useful operations on received data or replies with current sensor value. This function for client code is shown below, for simplicity in this example useful operation is just reply "echo" to sender:

```
static void
udp_handler(void)
{
    uip_ipaddr_copy(&client_conn_sync->ripaddr, &UIP_IP_BUF->srcipaddr);
    uip_udp_packet_sendto(client_conn_sync, "echo", strlen("echo"),
            &broadc_ipaddr, UIP_HTONS(UDP_SERVER_PORT_SYNC));
    uip_create_unspecified(&client_conn_sync->ripaddr);
}
```

When client or server receives new packet and if client or server should reply to sender, first the sender's address should be saved and after replying the address should be unspecified that it can receive unicasts from anyone else.

It is worth to mention that if server should send data packets to both multicast and unicast addresses it should keep one UDP connection (*struct uip_udp_conn*) for unicasts, and one for multicast on a separate ports. When client receive an incoming multicast packet, it will copy the packet's source address into the unicast destination address (the *ripaddr* field in *struct uip_udp_conn*), then send the packet, and finally reset the *ripaddr* so that it can receive unicasts from anyone again.

3.3 Summery

In this chapter 3 I described the requirements for hardware and software and which ones I choose. For the hardware I choose CC2650 multi-standard 2.4 GHz ultra-low power wireless MCU which is mounted on launchxl-cc2650 evaluation board. Additionally I used Raspberry Pi 3 model B with installed 6LoWPAN boarder router to access tested wireless sensor network from outside network (Internet). Also provided some pictures of single launchxl-cc2650 board as well as of complete test setup I used in this work.

In section reviewing software choice I describe operating system I decided to use – Contiki-OS and its advantages. Regarding operating system also was described its main functions and operating specifics such as routing and MAC layer. 6LowPAN boarder router running on Raspberry Pi was reviewed. For the firmware implemented I give a simplified example of RPL UDP client-server, on this firmware is based my realization of two synchronization protocols, which will be described later.

In next chapter I introduce reader to synchronization protocol and my realization of it in 6LoWPAN network.

4 Synchronization of end devices

The main motivation for this work is to realize mechanism for synchronizing sensor nodes in 6LoWPAN based wireless sensor network. In distributed wireless sensor networks maintaining clocks of the nodes' in such a way that they are very close to each other is one of the most complex problems, because of decentralized structure, latency of radio channel, big amount of devices to be synchronized, no defined standard, lossy network, lack of hardware support of existing synchronization protocols in most applications and etc.

In my master project I decided to use mechanisms of precision time protocol (PTP) version 2 defined in IEEE 1588-2008 standard, officially entitled "Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems" [19]. From my point of view this standard with its hierarchical master-slave architecture for clock distribution could be ideally adapted into 6LowPAN based networks, despite of the IEEE 1588-2008 standard describes the synchronization mechanism for wired network nodes, mainly using Ethernet. However I didn't find any IEEE802.15.4 wireless system on chip solution or low-power MCU with external transceiver that have IEEE1588 hardware support. For this reason the software implementation of precision time protocol is expected, however this reduces the synchronization accuracy to around 1 millisecond, while with hardware support the accuracy of 1 microsecond (or even sub-microsecond) is achievable.

In this chapter 4 in a brief and simplified manner I will review PTP version 2 including notes regarding deviations in my implementation from defined IEEE 1588-2008 standard. Also description of my implementations of PTPv2 will be described and explained.

4.1 PTPv2 message classes

PTP is client-server synchronization protocol, so to realize this protocol at least two devices are needed – master which have reference time and slave which should be synchronized to master's clock with most precise. This protocol is used to synchronize systems that include clocks of different precision, resolution and stability.

PTP is message-based protocol and messages are transported over UDP/IP. The PTP protocol defines event and general messages. The IEEE 1588-2008 standard defines in total ten messages which are divided into two sub-groups – event messages and general messages. Event messages are timed messages, accurate timestamp is generated at both transmission and receipt. General messages do not require accurate timestamps.

There are 4 event messages:

- Sync
- Delay_Req
- Pdelay_Req
- Pdelay_Resp

General messages are:

- Announce
- Follow_Up
- Delay_Resp
- Pdelay_Resp_Follow_Up
- Management
- Signaling

Announce message – sent by master to all slaves, contains information which is used to establish the synchronization hierarchy. According to information containing in this message, the slave clock can choose the best master clock (BMC). There is special algorithm how to choose BMC based on such values like accuracy, dispersion, rank, priority etc. For me BMC is not interest in the scope of this master project, because only one master clock is predefined – 6LoWPAN boarder router, so slave devices can't choose best master clock.

Management message – is used to query and update the PTP data sets, to customize a PTP system and also used for initialization and fault management. This message type is also omitted in my project.

Signalling message – is used for all other purposes.

Sync / Follow_Up, Delay_Req / Delay_Resp messages – used to generate and distribute the timing information needed to synchronize clocks by using the delay request-response mechanism. Follow_Up message could be omitted in one-step clock synchronization, but is mandatory in two-step clock synchronization.

Pdelay_Req / Pdelay_Resp and Pdelay_Resp_Follow_Up messages – used to measure the link delay between two clocks, this delay value is used to correct timing information contained in Sync and Follow_Up messages.

The slave is using only Delay_Req and Pdelay_Req request messages, the rest messages are sent by master clock, so master is required to provide almost all synchronization information. Messages are transmitted to multicast and unicast addresses. The IEEE 1588-2008 standard defines these multicast addresses, both IPv4 and IPv6 – 224.0.0.107 and FF02::6B for Pdelay_Req, Pdelay_Resp and Pdelay_Resp_Follow_Up messages; 224.0.1.129 and FF0x::181 for all other messages. Since devices I used don't have PTPv2 hardware support, in my protocol implementation I used IPv6 multicast address FF02::1 – all devices in network are required to receive messages from this multicast address (similar to broadcast messages in IPv4).

In my implementation of synchronization protocol for 6LoWPAN based network, multicast transmission is used for messages Sync and Follow_Up sent by master clock. The Delay_Req / Delay_Resp and Pdelay_Req / Pdelay_Resp and Pdelay_Resp_Follow_Up messages are used only in unicast transmission between master and slave. This will be explained later in chapter 4.4 describing my implementation of synchronization protocol.

4.2 PTPv2 device types

The IEEE 1588-2008 standard defines five basic types of PTP devices, which implement one or more sides of the protocol. These device types are:

- Ordinary clock
- Boundary clock
- End-to-end transparent clock
- Peer-to-peer transparent clock
- Management node

An ordinary clock has only one network connection. It can be either master providing reference time or slave that is a destination for synchronization time reference.

A boundary clock is like ordinary clock but with multiple network connections. One of this connection is in slave mode, receiving time from master clock. Other connections (might be more than one) acting as master providing synchronization time to downstream slaves (it might be ordinary or boundary clock). So boundary clock receives synchronization on slave port, corrects own time, generate new synchronization message and finally transmit this message to slaves from master port.

An end-to-end transparent clock and peer-to-peer transparent clock are devices that pass synchronization message, but updating synchronization time with corrected time for message spent traversing the network. This mechanism improves accuracy by compensating the traverse latency across the network.

A management node is a device that may have one or more network connections. It may be combined with any of the device types described above.

It is also worth to mention about grandmaster clock. This is the root of timing reference. Grandmaster always acts as a master, it has precise oscillator and can get time for example from GPS. The grandmaster is only one in network. In my implementation the 6LoWPAN boarder router is a grandmaster, it transmits synchronization messages to all the slaves.

4.3 PTPv2 simple master-slave clock hierarchy

One of many possible PTP master-slave hierarchies is illustrated in Figure 13. In this example ordinary clock 1 is grand master and located at the root of hierarchy. Boundary clock 1 is directly connected to grand master and receives synchronization time on its

slave port. Other ports of boundary clock 1 are acting as masters providing synchronization time to downstream slaves – ordinary clock 2, ordinary clock 3 and to boundary clock 2. Boundary clock 2 is similar to boundary clock 1, provides synchronization time from its master ports to downstream slaves – ordinary clock 4, ordinary clock 5 and ordinary clock 6. This simple PTP master-slave hierarchy could be easily extended with other boundary clocks and ordinary clocks, also it is possible to include transparent clocks.

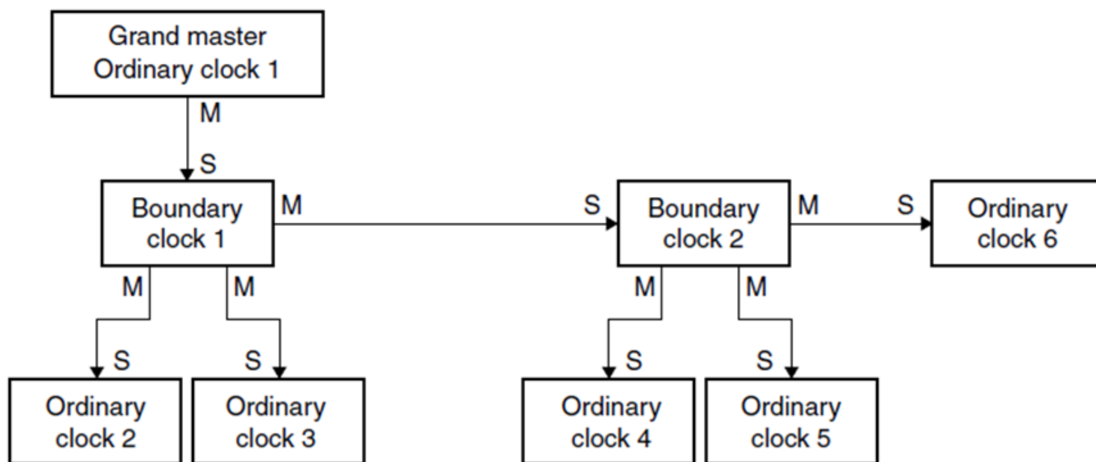

Figure 13. PTPv2 simple master-slave clock hierarchy.

My implementation of precision time protocol has even simpler master-slave clock hierarchy than shown above. For testing purposes and because of reduced amount of devices, the clock hierarchy includes ordinary clock 1, acting as grand master (reference time was its own general purpose timer) and 3 ordinary clocks which were synchronized against this grand master clock, however software support for enabling boundary clocks were implemented. So in future it will be good to test more complex clock hierarchy for synchronizing nodes in 6LoWPAN based network.

4.4 Implementation of modified PTPv2

In this chapter I would like to describe my implementation of PTPv2, of course it is not a full implementation of the IEEE 1588-2008 standard, since it was developed for wired networks, specifically Ethernet. My goal is to use the core concepts of precision time protocol and apply them to wireless sensor network based on 6LoWPAN standard. I think

that my implementation is better to be termed as modified PTP since it is not fully conform the specified standard.

The simplest implementation of PTP is an application at the top of the network protocol stack and timestamps are generated at the application level. This approach brings error in timestamp caused by floating delay. When timestamp is generated at the application level, the time to go down the stack to actually send out synchronization message, is not constant, depending on system workload. The same floating delay is introduced at the recipient side – non constant time to propagate synchronization data from physical layer up the stack to application layer. These errors are typically in the hundred microseconds to milliseconds range depending on the system. This is what makes software realization of PTP not such accurate as with hardware support, the difference is about 1000 times (1 microsecond precision with HW support and 1 millisecond precision without HW support). Not only this error has bad effect on synchronization accuracy, the following is limiting the achievable accuracy of PTP system:

- The delay fluctuation in the protocol stacks of clocks
- The delay asymmetry
- The delay fluctuation in network components
- Timestamping accuracy
- Stability issues

The pattern of basic synchronization message exchange is illustrated in Figure 14.

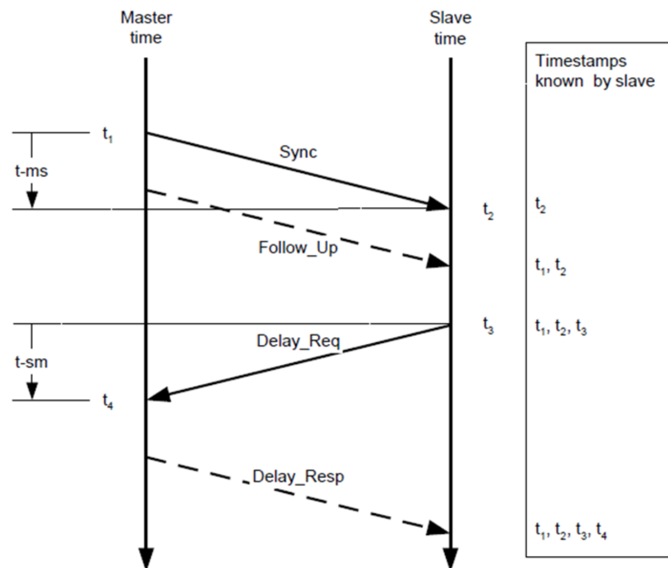

Figure 14. PTP message exchange.

The synchronization process is divided into two steps: first is slave's clock offset correction and secondly determining the message propagation delay.

Synchronization process begins with Sync message sent by master clock to all slaves to multicast address. With a Sync message transmission master clock saves time t1, when message is sent. In my implementation I use two step mode, so immediately after Sync message, master sends Follow_Up message containing timestamp of t1, this message is also sent to multicast address. When the slave receives Sync message it generates timestamp t2. The slave then takes time value in Follow_Up message and subtracts the timestamped value t2, resulting a calculated offset (here is no difference if I calculate offset by subtracting t2 from t1 or t1 from t2). The calculated offset is now added to current slave time resulting a new clock time. Master and slave clocks are not yet completely synchronized because the network delay is not taken into account.

After slave clock calculated its offset regarding master clock, starts the second step of synchronization process. Now slave sends out Delay request message to the master, also by sending out Delay_Req message, slave generates and saves timestamp t3 with its local time. The master clock receives this messages, generates its local timestamp t4 and sends out the value of t4 in Delay response message. This Delay_Resp message is sent out in unicast mode, the destination address is the slave that sent delay request message. When slave clock receives delay response message it can calculate the network delay by

subtracting the time it sent the delay request message (t3 timestamp) from the time the master received the delay request message (t4 timestamp). However the delay was measured by sending two messages, so to obtain correct one-way message propagation delay, the last calculated value (t4 – t3) should be divided by 2 and now slave clock can finally add this delay value to its current time, giving the synchronized clock between master and slave, of course with limited accuracy.

The internal clocks of the different devices will inevitably drift over time. To compensate for this, the master clock will periodically send out Sync message and the synchronization process will be started over again. The synchronization periodicity shouldn't be very frequent to not over flood the network. With a described synchronization process all slave clocks will be synchronized to master clock one by one after they receive two multicast messages – Sync and Follow_Up.

Additionally to already described synchronization process I implemented another one, pretty much the same but slightly modified. Figure 15 illustrates modified synchronization message exchange.

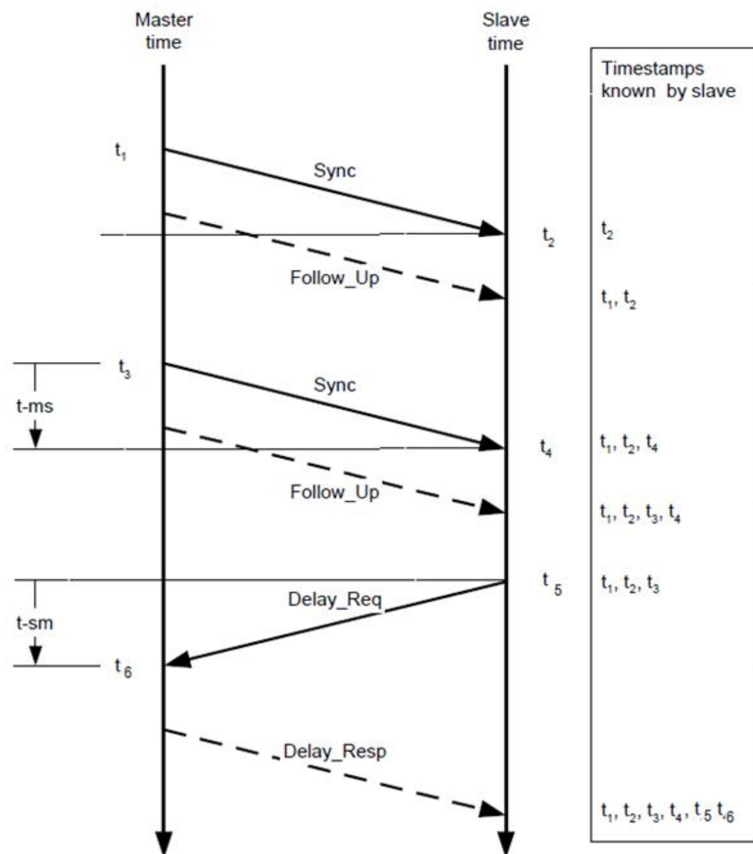

Figure 15. Modified PTP message exchange.

Comparing modified PTP message exchange with message exchange described before, one additional pair of multicast messages Sync and Follow_Up are sent by master clock. This is done to achieve better accuracy which is limited with network delay fluctuations. To correct for the message transmission delay, the slave additionally uses a second pair of Sync message and Follow_Up message with its corrected clock to calculate the master-to-slave delay. The second set of messages is necessary to account for variations in network delays. So if there will be difference between timestamps t3 and t4, the slave will correct its internal clock value by this difference. The rest of synchronization process and message exchange is described above and it is similar to basic PTP message exchange.

Below is Figure 16 illustrating code flowchart of master clock. The code is similar for both described PTP synchronization processes, except additional pair of Sync and Follow_Up messages in case of second described implementation. On the Figure 19 additional Sync/Follow_Up pair is presented.

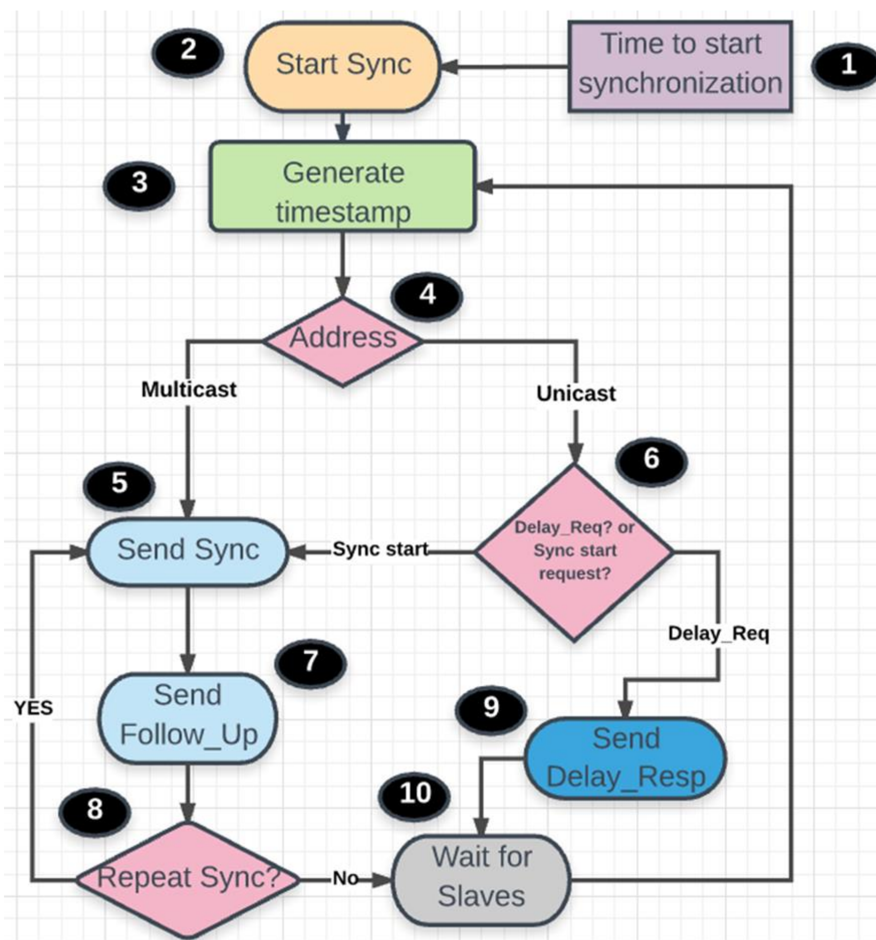

Figure 16. Master clock code flowchart.

The synchronization process starts by timer 1). In step 3) timestamp is generated and in the next step 4) is taken decision whether to send synchronization message to multicast or if master received request from slave, to unicast address. Address is copied to created UDP connection structure, which mainly contains addresses, ports and data to send. If it is multicast address then master should send the pair of Sync / Follow_Up messages (in case of first described implementation only one pair is sent). When master clock received request from slave clock it should do get know, step 6), which type of request it is. Normally slave should send to master clock only delay requests, but during tests it came up that rarely one 6LoWPAN node measured too long delay (up to 2 seconds) between Delay_req and Delay_resp, in this situation node will ask to start synchronization from step 5), but in unicast mode. If it was Delay_req message then master replies 9) with delay response message. Finally in step 10) master clock starts to wait messages from slaves (delay request or Sync repeat), also periodic timer 1) could start synchronization process from the beginning.

Below is Figure 20 illustrating code flowchart of slave clock. As in case of previously illustrated master clock code flowchart, this code is also similar for both described PTP synchronization processes, except additional pair of Sync and Follow_Up messages in case of second described implementation. On the Figure 17 additional Sync/Follow_Up pair is shown.

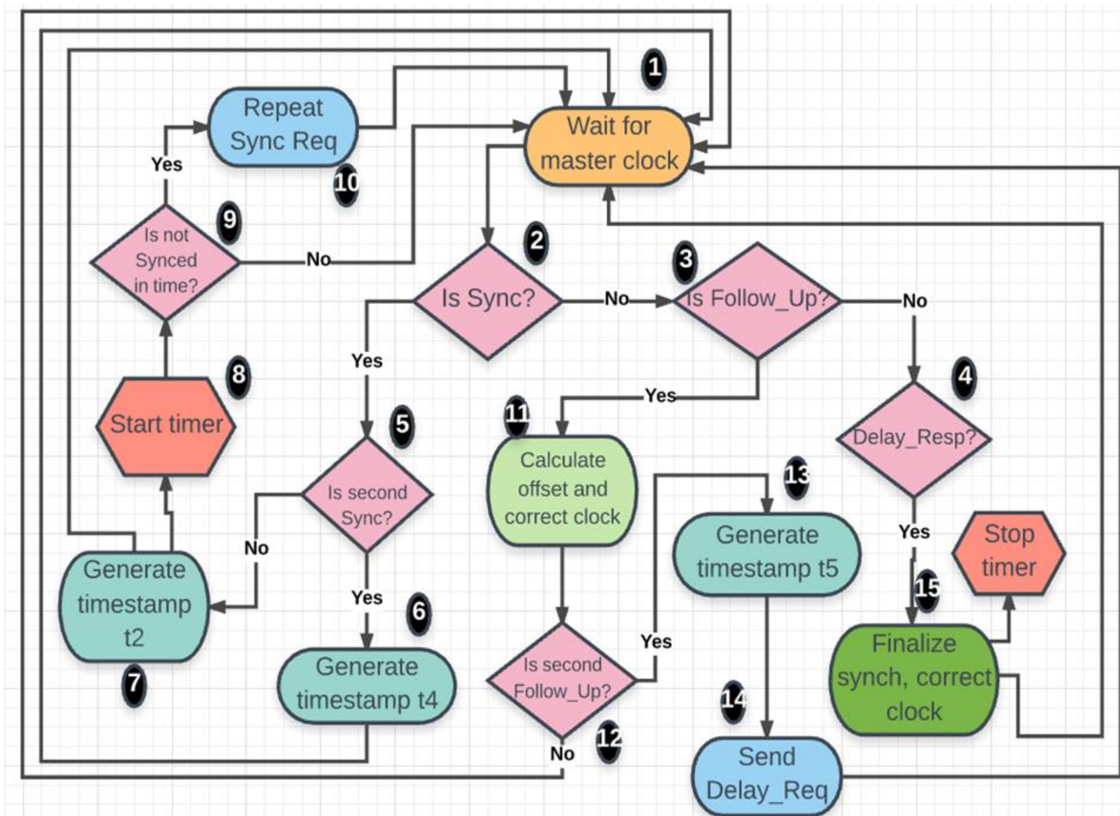

Figure 17. Slave clock code flowchart.

The slave clock continuously waits for PTP messages from master clock. In the steps 2), 3) and 4) slave clock gets know the type of received message. If it is first Sync message then slave clock in step 7) generates timestamp t2, starts timer and continues to wait messages. Also with reception of first Sync message, slave clock in step 8) starts its timer, which will interrupt and request synchronization started with Sync message, but in unicast mode, when synchronization not completed during defined time. If it was second Sync message then in step 6) generates timestamp t4 and also continues to wait messages from master clock. When received message is Follow_up, slave clock calculates offset and corrects clock in step 11), and also slave clock checks whether it is second or first Follow_Up message. In case it is first Follow_Up message – continues to wait messages from master, if second – generates timestamp t5 in step 13), sends out delay request message in step 14) and jumps to wait next message. Finally when Delay_Resp message is received, slave clock calculates the network delay and corrects current local time. Also In step 15) timer, measuring time to complete synchronization is stopped and slave clock jumps to wait messages for net synchronization process.

In the next chapter 5 will be described testing process of two implemented realizations of modified synchronization protocol defined in IEEE 1588-2008 standard. Also all available measurements will be presented.

4.5 Summery

In this chapter the basics of precision time protocol version 2 defined in IEEE 1588-2008 standard were described. The description includes message classes, device types and master-slave hierarchy. My 2 versions of modified realization of this standard also was described including code flowcharts for both master clock and slave clock operation. In the next chapter I will present test and measurement results for both implementations of synchronization protocol.

5 Measurements results

One of the main targets of my thesis work is to evaluate the capability of 6LoWPAN based wireless sensor network. In this chapter I describe methods, processes and types of measurements and tests I made. In chapter 5.1 acquired results for synchronization are shown and studied, in chapter 5.2 results of data transfer tests are shown and studied.

The real field wireless mesh sensor network may consist of tens and hundreds of nodes which are spatially distributed, however I have a limited amount of devices and space. For evaluating of synchronization protocols I developed, the test setup include the following components. Three end-nodes (RPL-UDP clients) wirelessly connected (rooted) to one DAG root (RPL-UDP server). All devices are same TI CC2650 LaunchPad(s), they are spatially distributed from each other in distance of 10-100 cm. The devices are USB-powered from one source. The nodes (clients) are numbered as node1, node2 and node3, the devices doesn't interchange their numbering during taking measurements (also same numbering for nodes is constant throughout the thesis). The server is interchangeably named as server, master, and UDP server.

One type of synchronization measurements were done by using oscilloscope Owon DS 5032E bandwidth 30 MHz, sampling rate 500 MSa/s. Another synchronization test was done with additional MSP430 LaunchPad [20] which was used to generate pulses every 2 seconds, the synchronized timers values were sent to PC via serial interface and later calculations were done in Excel. Measurements were done for both synchronization protocols under equal conditions.

For testing of data transfer capability, one of available CC2650 LaunchPad was used as IEEE 802.15.4 packet traffic sniffer and Wireshark [21] [22] to capture and correctly show 6LoWPAN traffic.

In the following subchapters measurements described in detail and result are shown. Note for reader: in the following sub-chapters terms "version 1", "version 2", "PTPv1" and "PTPv2" are referring to 2 synchronization versions that I implemented, which were described in chapter 4.

5.1 Oscilloscope measurements

In this type of measurement GPIO Pin 15 (same on server and clients) was configured as output and was toggled every 500 ms on GPT2A TIMER value match interrupt. The same timer, was synchronized to server's GPT2A timer. This timer is configured as periodic, count-up, 32 bit full-width and no pre-scaler. The interrupt on timer value match was enabled and handled in same interrupt service routine as for timer overflow. Every time timer value match interrupt occurs the DIO 15 is toggled and timer match value is increased to toggle pin again after next 500 ms. On timer overflow same process starts from zero timer value. There is no reason to toggle pin more frequent. All traffic except synchronization data is avoided during measurements.

The following code snippet shows the GPT2A interrupt service routine.

```
void
ISR_GPT2A()
{
const uint32_t LOAD_VAL = timer_load_get(GPT2A);

if(TIMER_TIMA_MATCH == timer_int_status(GPT2A, 1))
{
if((timer_value_get(GPT2A) + SET_MATCH_VAL) < LOAD_VAL)
timer_match_set(GPT2A, (timer_value_get(GPT2A)+SET_MATCH_VAL));
else
timer_match_set(GPT2A, SET_MATCH_VAL);

gpio_toggle_dio(DIO_15);
timer_int_clear(GPT2A, TIMER_TIMA_MATCH);
}
if(TIMER_TIMA_TIMEOUT == timer_int_status(GPT2A, 1))
{
//gpio_toggle_dio(DIO_15);
leds_toggle(LEDS_RED);
timer_int_clear(GPT2A, TIMER_TIMA_TIMEOUT);
}

// I need to make sure event is cleared before returning from the ISR.
// Allowing cleared event to propagate through any synchronizers.
__asm__ ("nop");
__asm__ ("nop");
}
```

All synchronization precision measurements are done with reference to master clock, meaning that one oscilloscope probe was attached to DIO15 of cc2650 LaunchPad acting

as server, another probe was attached to same pin on node1, then node2 and then node3. Measurements between nodes were done during testing, but not included in this work, since they are in same range as to measurements taken with reference to server. The time difference between node's pin rising edge and server's pin rising edge show the precession of synchronization. The less time difference (Δt) the more precisely timers are synchronized. Besides the measuring Δt right after synchronization on first rising edge, 5 additional measurement points after synchronization were taken: 30 s, 60 s, 180 s, 240 s and 300 s after synchronization. This is done to observe jitter between different timer clocks and to estimate how frequently I need to send synchro signal to keep desirable synchronization precision. Moreover the time spent for each synchronization was also measured.

For every node 10 synchronization results were taken. After each attempt all nodes and server were restarted. By button press on LaunchPad acting as UDP server, the synchronization process was initialized and end-devices were synchronized.

Both versions of developed synchronization protocols were tested in the same way. Below the acquired results are present in separate tables for each node and for each synchronization protocol. The differences in protocol versions are described in chapter 4.

Results of synchronization protocol version 1:

Table 1. Results of synchronization protocol version 1, node 1.

Node1							
Num.	Δt [us]	Time to SYNC [ms]	Δt [us] after 30s	Δt [us] after 60s	Δt [us] after 180s	Δt [us] after 240s	Δt [us] after 300s
1	402	4.789	426	452	544	592	640
2	204	33.745	232	252	348	388	444
3	320	4.8	388	412	508	556	612
4	556	91.148	584	608	724	772	836
5	384	7.399	408	463	544	596	652
6	358	4.846	379	411	508	552	614
7	312	22.453	345	392	496	538	612
8	367	47.104	393	430	523	564	625
9	391	22.269	337	378	471	539	619
10	287	23.474	318	346	452	461	527
Average	358	26.2027	381	414	512	556	618
Median	363	22.361	384	412	508	554	617

The results for node1, in Table 1, show that synchronization precision is better than 1 millisecond, the average time difference for 10 measurements is 358 microseconds. Also from table results I can observe the difference between server's and client's clocks after some time. In average the timers are desync on approximately 60 microseconds in every 1 minute, however even after 5 minutes the Δt is still less than 1 millisecond.

Here are results for node2 in above Table 2, the precision of synchronization is also better than 1 millisecond, however the crystal oscillator of node2 include its error and already after 3 minutes node2 and server are more than 1 millisecond out of sync. Desync between client and server is about 200 microseconds per every minute. In this example to achieve time difference of less than 1 millisecond between devices, I need to send synchro signal at least every 2 minutes.

Table 2. Results of synchronization protocol version 1, node 2.

Node2							
Num.	Δt [us]	Time to SYNC [ms]	Δt [us] after 30s	Δt [us] after 60s	Δt [us] after 180s	Δt [us] after 240s	Δt [us] after 300s
1	480	16.771	600	692	1084	1296	1500
2	444	24.595	548	648	1068	1240	1452
3	448	16.811	552	664	1060	1264	1462
4	468	16.779	584	680	1074	1266	1484
5	430	5.325	541	634	1014	1200	1397
6	457	11.654	568	691	1086	1302	1506
7	445	17.893	554	667	1058	1261	1459
8	463	16.52	579	676	1047	1249	1460
9	478	23.547	593	688	1081	1293	1494
10	435	14.392	536	624	1011	1195	1373
Average	455	16.4287	566	666	1058	1257	1459
Median	453	16.775	561	672	1064	1263	1461

Below in Table 3 results of synchronization measurements for node3. As in case of node1 and node2 the precision of synchronization is better than 1 millisecond, with average for 10 measurements 490 microseconds. Desync between node3 and server is about 70 microseconds per 1 minute. In case of node3 to achieve time difference less than 1 millisecond the device should be synchronized every 5 minutes.

Table 3. Results of synchronization protocol version 1, node 3.

Node3							
Num.	Δt [us]	Time to SYNC [ms]	Δt [us] after 30s	Δt [us] after 60s	Δt [us] after 180s	Δt [us] after 240s	Δt [us] after 300s
1	448	16.339	486	516	648	712	780
2	457	16.794	488	518	658	722	788
3	470	17.804	508	542	676	744	812
4	556	4.777	588	620	756	832	890
5	481	33.489	517	551	694	761	821
6	479	10.952	521	563	698	769	834
7	523	27.23	554	594	713	792	846
8	511	11.687	546	587	701	774	829
9	466	30.269	502	548	690	765	833
10	505	14.841	553	604	745	827	884
Average	490	18.4182	526	564	698	770	832
Median	480	16.5665	519	557	696	767	831

Results of synchronization protocol version 2:

Table 4 below with results of measurements for node1 with protocol version 2 shows that time difference between node1 and server right after synchronization is under 200 microseconds. After 5 minutes the time difference between devices is 468 microseconds in average for 10 measurements.

Table 4. Results of synchronization protocol version 2, node 1.

Node1							
Num.	Δt [us]	Time to SYNC [ms]	Δt [us] after 30s	Δt [us] after 60s	Δt [us] after 180s	Δt [us] after 240s	Δt [us] after 300s
1	31	10.995	58	80	185	236	288
2	30	10.908	66	93	191	236	283
3	61	10.944	88	111	208	254	306
4	198	68.771	230	260	404	478	566
5	169	22.942	191	226	402	484	568
6	192	33.732	225	294	642	726	808
7	177	25.771	204	234	406	494	578
8	158	18.73	190	225	324	371	421
9	147	22.986	182	219	309	367	414
10	164	4.789	201	226	323	378	447
Average	133	23.0568	164	197	339	402	468
Median	161	20.836	191	226	324	375	434

Below in Table 5 results show synchronization precision of node2 using protocol version 2. The time difference between client and server right after synchronization is about 117

microseconds. After 5 minutes time difference is slightly above 1 millisecond, so to have Δt constantly less than 1 millisecond node2 should be synchronized roughly every 4 minutes.

Table 5. Results of synchronization protocol version 2, node 2.

Node2							
Num.	Δt [us]	Time to SYNC [ms]	Δt [us] after 30s	Δt [us] after 60s	Δt [us] after 180s	Δt [us] after 240s	Δt [us] after 300s
1	99	18.193	218	316	696	867	1104
2	3	10.903	101	180	580	780	978
3	180	47.248	280	381	758	967	802
4	80	22.985	82	121	498	894	1085
5	108	25.496	222	329	728	932	1132
6	110	22.873	130	230	602	806	1018
7	182	33.483	292	388	784	986	1185
8	112	33.437	220	328	732	928	1120
9	178	10.903	287	378	784	984	1180
10	117	10.938	178	267	668	870	1066
Average	117	23.6459	201	292	683	901	1067
Median	111	22.929	219	322	712	911	1095

The results for node3 are provided in Table 6 below. The synchronization precision is better than 200 microseconds. After 5 minutes time difference between node3 and server is about 500 microseconds, meaning if I send synchro signal at least every 5 minute node3 will be synchronized to server clock with precision under 1 microsecond.

Table 6. Results of synchronization protocol version 2, node 3.

Node3							
Num.	Δt [us]	Time to SYNC [ms]	Δt [us] after 30s	Δt [us] after 60s	Δt [us] after 180s	Δt [us] after 240s	Δt [us] after 300s
1	110	29.917	161	183	318	388	455
2	124	22.886	172	199	333	402	470
3	196	10.864	233	264	404	471	542
4	198	41.527	252	290	423	492	558
5	9	23.783	45	78	216	286	353
6	172	18.69	224	261	401	469	534
7	184	55.472	236	273	408	477	549
8	135	21.354	184	229	375	441	501
9	166	28.355	207	239	382	458	522
10	151	10.948	198	241	386	463	537
Average	145	26.3796	191	226	365	435	502
Median	159	23.3345	203	240	384	461	528

Below I give summery Table 7 combining all results separately for both protocols. It will simplifier for readers to have a performance of synchronization precision for both developed protocols and to see the differences.

Table 7. Summery tables combining all results separately for both protocols.

Version1	Δt [us]	Time to SYNC [ms]	Δt [us] after 30s	Δt [us] after 60s	Δt [us] after 180s	Δt [us] after 240s	Δt [us] after 300s
Average	434	20	491	548	756	861	970
Median	453	17	529	575	700	771	834

Version2	Δt [us]	Time to SYNC [ms]	Δt [us] after 30s	Δt [us] after 60s	Δt [us] after 180s	Δt [us] after 240s	Δt [us] after 300s
Average	131	24	185	238	462	580	679
Median	149	23	200	237	404	478	554

From the Table 7 it is seen that precision of synchronization is better in case of protocol version 2. However both protocols give precision under 1 millisecond. To have a time difference between all nodes and server during devices' uptime the synchronization process should be initiated at least every 2 minutes in case of protocol version 1 and every 4 minutes in case of second protocol (to gather these times you need to refer to detailed tables with results).

5.2 Synchronization precision measurements with external signal

Another type of measurement to evaluate the synchronization precision of both developed protocols is described in this chapter. In this test I used MSP430 LaunchPad to generate pulses every 2 seconds. All three nodes and server were configured to generate interrupt when on cc2650 LaunchPad pin 21 rising edge is detected. When pulse is detected the synchronized timer value (and server's timer value) is printed out on serial port. All three nodes' and server input pin 21 connected to breadboard, MSP430 output pin generating pulses also connected to same line on breadboard. With this approach I know that pulses are detected by devices in exactly same time point. No other tasks were executed on devices of interest, only once before measurement started nodes were synchronized and approximately 40-50 seconds after MSP430 board started to generate pulses by pushing

the button on board. 10 measurements during 20 seconds were taken, so this will show how nodes' timers differ from master timer during first minute after synchronization. Printed results were captured with PuTTY [23] and later calculated in Excel.

Below results for both protocols are presented in Table 8 and Table 9, where present values of 32 bit timers. Timers are not pre-scaled (no such possibility in cc2650 for full-width mode timers), so its clocks are as MCU's – 48 MHz, calculated time differences are shown in microseconds and in timer ticks.

Table 8. Measurement results taken with external signal, for version 1.

PTPv1										
	Master timer	Node1 timer	Δt [ticks]	Δt [us]	Node2 timer	Δt [ticks]	Δt [us]	Node3 timer	Δt [ticks]	Δt [us]
Edge0	2275053187	2275037503	15684	327	2275031333	21854	455	2275028889	24298	506
Edge1	2446359002	2446342886	16116	336	2446336986	22016	459	2446334903	24099	502
Edge2	2617607635	2617589989	17646	368	2617585459	22176	462	2617583735	23900	498
Edge3	2788872915	2788855578	17337	361	2788850578	22337	465	2788849215	23700	494
Edge4	2960123413	2960105917	17496	365	2960100917	22496	469	2960099914	23499	490
Edge5	3131380823	3131362166	18657	389	3131358166	22657	472	3131357523	23300	485
Edge6	3302649508	3302629998	19510	406	3302626692	22816	475	3302626409	23099	481
Edge7	3473918519	3473898544	19975	416	3473895544	22975	479	3473895620	22899	477
Edge8	3645187496	3645167361	20135	419	3645164361	23135	482	3645164797	22699	473
Edge9	3816450757	3816429972	20785	433	3816427462	23295	485	3816428257	22500	469
AVG				382			470			487

In table 8 results for protocol version 1, it can be seen that time difference between nodes and timer does not exceed 500 microseconds in the end of first minute after synchronization. The clocks deviate from masters on 3-10 microseconds in every 2 seconds.

Table 9. Measurement results taken with external signal, for version 2.

PTPv2										
	Master timer	Node1 timer	Δt [ticks]	Δt [us]	Node2 timer	Δt [ticks]	Δt [us]	Node3 timer	Δt [ticks]	Δt [us]
Edge0	1950535596	1950548178	12582	262	1950555068	19472	406	1950512002	23594	492
Edge1	2121837283	2121850012	12729	265	2121857044	19761	412	2121813930	23353	487
Edge2	2293098506	2293111856	13350	278	2293118559	20053	418	2293075755	22751	474
Edge3	2464353971	2464367556	13585	283	2464374313	20342	424	2464331703	22268	464
Edge4	2635590889	2635604980	14091	294	2635611520	20631	430	2635568702	22187	462
Edge5	2806798285	2806812642	14357	299	2806819206	20921	436	2806776240	22045	459
Edge6	2978044604	2978058999	14395	300	2978065818	21214	442	2978022663	21941	457
Edge7	3149290414	3149304983	14569	304	3149311918	21504	448	3149268695	21719	452
Edge8	3320560819	3320575684	14865	310	3320582612	21793	454	3320539228	21591	450
Edge9	3491819585	3491834787	15202	317	3491841668	22083	460	3491798208	21377	445
AVG				291			433			464

In above table results for synchronization protocol version 2 are present. Time difference between nodes and timer as in case of protocol version 1 does not exceed 500 microseconds in the end of first minute after synchronization and even better. The clocks deviate from masters on 3-10 microseconds in every 2 seconds.

This test was additionally done to prove results acquired in chapter 5.1, where measurements were taken by oscilloscope right after synchronization and at some points after. These results show that time difference between nodes and server are less than 1 millisecond and even under 500 microsecond. I think that achieved results are good and believe that synchronization protocols might be improved to achieve better precision.

5.3 Time to synchronize

One measurement which I not described yet is time to sync. This time was measured with timer GPT3A, which was started when first multicast synchro signal is reached to node's application layer software and stopped also in application layer when last unicast synchro signal received and timer to synchronize is adjusted with new value. In this case average or median values doesn't tell much, since after nodes receive multicast synchro signal they start to talk to server in unicast way, and they start to send data exactly in same time, so concurrency and/or collisions occur. This means that only 1 node at time moment is able to send data frame over IEEE 802.15.4 network. There always will present non-constant time between first and last synchronized node. And this time almost linearly increase with adding new nodes into network.

In this chapter I bring results of measured time to synchronize. In tables you will find 10 measurements for each protocol. In every attempt time to sync is taken for all three nodes, each node send its measured value to PC via serial interface.

As it could be seen from Table 10 below, the time to sync is non deterministic and in most attempts pretty much same for both protocols. It is easy to explain – despite that synchronization protocol version 2 uses 4 times more multicast synchro frames, the most time spent on synchronization is used by unicast frames where concurrency and collisions are met, and this is equal issue for both protocols. In Table 10 during attempt 5 in ver.1 happened situation when all three nodes did not complete synchronization during defined time limit and nodes separately were asking server for new synchro signals. It is not

mandatory that all three nodes will be not synchronized with defined time limit, during tests I have seen that 1 or 2 nodes were not synchronized during time limit, however this does not have influence on precision. Unfortunately I don't have statistics how often synchronization time is out of limit, but this was quite rare situations.

Table 10. Time to synchronization, version 1 and version 2

Version 1		
SYNC num.	Node	Time to SYNC [ms]
1	Node1	4.791
1	Node2	25.947
1	Node3	17.804
2	Node1	4.814
2	Node2	25.947
2	Node3	52.544
3	Node1	4.781
3	Node2	24.543
3	Node3	10.791
4	Node1	7.574
4	Node2	18.716
4	Node3	34.781
5	Node1	336.951
5	Node2	353.69
5	Node3	384.84
6	Node1	6.845
6	Node2	11.359
6	Node3	23.64
7	Node1	17.28
7	Node2	14.574
7	Node3	4.814
8	Node1	16.804
8	Node2	11.826
8	Node3	4.807
9	Node1	10.952
9	Node2	14.692
9	Node3	23.128
10	Node1	4.805
10	Node2	19.646
10	Node3	9.979

Version 2		
SYNC num.	Node	Time to SYNC [ms]
1	Node1	30.721
1	Node2	38.761
1	Node3	10.913
2	Node1	16.57
2	Node2	28.61
2	Node3	43.956
3	Node1	11.28
3	Node2	20.478
3	Node3	46.601
4	Node1	10.94
4	Node2	20.507
4	Node3	30.766
5	Node1	10.949
5	Node2	38.878
5	Node3	28.298
6	Node1	22.965
6	Node2	38.623
6	Node3	10.968
7	Node1	48.105
7	Node2	16.3
7	Node3	55.562
8	Node1	18.72
8	Node2	35.307
8	Node3	28.25
9	Node1	25.829
9	Node2	22.64
9	Node3	33.359
10	Node1	25.771
10	Node2	16.778
10	Node3	10.911

In my test setup I have only 3 nodes and time to sync is not critical, but with every new node in network the estimated time to sync for last node increases on approximately 10 milliseconds. So if 6LoWPAN network for example contains 10 nodes, theoretical estimated time to sync for last node is about 100 milliseconds in best case situation. But as I can see from my results where 3 nodes in network, time is non deterministic and collision may lead to situation when synchronization retransmission are needed due to reached synchronization time limit. This might lead to big problems when some tens of nodes simultaneously will send unicast frames to the server asking for synchro. In a worst case scenario might happen situation when no one node will be able to synchronize. One of theoretical solutions for this problem I think is that every node should wait some random time before sending unicast frame to server. This should help to relieve load from CSMA and less collisions will occur. Of course this is only theory not supported by real tests.

5.4 6LoWPAN data transfer capability test

In this chapter I describe how test of 6LoWPAN data transfer capability were done and show the results I have. In this test, one of available CC2650 LaunchPad was used as IEEE 802.15.4 packet traffic sniffer. So I have 1 6LoWPAN device acting as server and 2 nodes which continuously and simultaneously send UDP data to server. Wireshark [21] was used to correctly show 6LoWPAN traffic captured with sniffer.

The firmware for sniffer was taken from Contiki github, few modifications were done to match target hardware CC2650 LaunchPad. To capture and proceed frames captured by sniffer, free python script “Sensniff” - Live Traffic Capture and Sniffer for IEEE 802.15.4 networks [22] was used. Sensniff helps to do real traffic capture and further data analysis. This script is available only for *nix platforms so virtual machine with Ubuntu OS was used. CC2650 LaunchPad acting as sniffer was connected to virtual Ubuntu via USB bus. Wireshark on Ubuntu was configured to capture data from Sensniff and correctly represent 6LoWPAN data.

Tests were done in following way: server send 1 multicast frame, 2 nodes receive this frame and simultaneously start to send UDP data packets. Each time server receives packet it reply with hardware-level acknowledgment frame ACK sent by the radio interface. In total were done 39 tests – data was sent with different frequency and different

data length. In all cases 2 nodes were sending with equal time periodicity and equal data length. Below is column chart with gathered results.

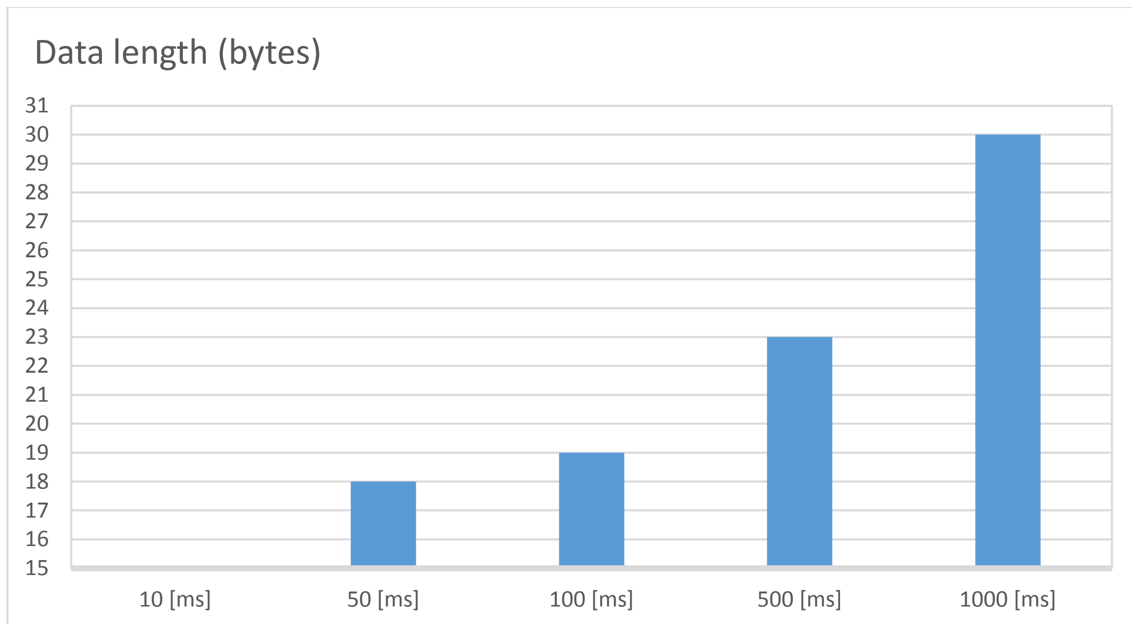

Figure 18. Data transfer capability test results.

In column chart shown above the results of 6LoWPAN data transfer capability tests are presented. Here under X-axis are time periodicity of sending data by 2 nodes simultaneously. Under Y-axis data length of each packet.

The maximum data length that could be sent simultaneously by 2 nodes every 50 milliseconds is 18 bytes; every 100 milliseconds – 19 bytes; every 500 milliseconds – 23 bytes; every 1000 milliseconds – 30 bytes. When I tried to transfer data every 10 millisecond the network was overloaded and 1 node was never able to send its data (despite the data length) to server, so I consider it as 6LoWPAN impossibility to handle data transfer from multiple nodes while they transfer data simultaneously every 10 milliseconds. Below is snapshot from Wireshark when IEEE 802.15.4 is overloaded and data is not received by server.

No.	Time	Source	Destination	Protocol	Length	Info
1017	52.491241000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1018	52.491642000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1019	52.495790000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1020	52.500550000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1021	52.501056000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1022	52.501343000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1023	52.539651000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1024	52.540794000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1025	52.541858000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1026	52.544733000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1027	52.545832000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1028	52.556182000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1029	52.556664000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1030	52.599456000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1031	52.599698000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1032	52.599926000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1033	52.600127000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1034	52.600401000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1035	52.600522000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1036	52.600621000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1037	52.600719000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1038	52.652674000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1039	52.654031000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination
1040	52.655119000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	51	Source port: 61617 Destination

```
▸ Frame 1033: 51 bytes on wire (408 bits), 51 bytes captured (408 bits) on interface 0
▸ IEEE 802.15.4 Data, Dst: TexasIns_00:0d:6a:dc:87, Src: TexasIns_00:0d:54:3d:82
▸ 6LoWPAN
▸ Internet Protocol Version 6, Src: fe80::212:4b00:d54:3d82 (fe80::212:4b00:d54:3d82), Dst: fe80::212:4b00:d6a:dc87 (fe80::212:4b00:d6a:dc87)
▸ User Datagram Protocol, Src Port: 61617 (61617), Dst Port: 61618 (61618)
▸ Data (22 bytes)
```

Figure 19. Two nodes fail to simultaneously send 22 data bytes every 100 milliseconds.

In Figure 19 is seen how nodes are trying to send 22 data bytes every 100 milliseconds, but server never replies. The total packet size including compressed header is 51 bytes. The node with link-local address fe80::212::4b00:d54:3d82 is node2 in this thesis; node with link-local address fe80::212::4b00:d5e:d403 – is node3; and server's address is fe80::212::4b00:d6a:dc87.

Below I want to show Wireshark snapshot when nodes were able to continuously send data and server acknowledges on received frames. In this example 19 data bytes were sent every 500 milliseconds. The total packet size including compressed header is 48 bytes.

No.	Time	Source	Destination	Protocol	Length	Info
647	51.673953000			IEEE 802.15.4	5	Ack
648	51.732058000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
649	51.732634000			IEEE 802.15.4	5	Ack
650	51.732869000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
651	51.733176000			IEEE 802.15.4	5	Ack
652	51.851422000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
653	51.851608000			IEEE 802.15.4	5	Ack
654	51.851711000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
655	51.851832000			IEEE 802.15.4	5	Ack
656	51.971449000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
657	51.971934000			IEEE 802.15.4	5	Ack
658	51.972152000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
659	51.972550000			IEEE 802.15.4	5	Ack
660	52.030930000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
661	52.031310000			IEEE 802.15.4	5	Ack
662	52.031424000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
663	52.031527000			IEEE 802.15.4	5	Ack
664	52.151148000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
665	52.153562000			IEEE 802.15.4	5	Ack
666	52.154749000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
667	52.157086000			IEEE 802.15.4	5	Ack
668	52.270564000	fe80::212:4b00:d5e:d403	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
669	52.271037000			IEEE 802.15.4	5	Ack
670	52.271465000	fe80::212:4b00:d54:3d82	fe80::212:4b00:d6a:dc87	UDP	48	Source port: 61617 Destination por
671	52.271815000			IEEE 802.15.4	5	Ack

```
Frame 656: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface 0
IEEE 802.15.4 Data, Dst: TexasIns_00:0d:6a:dc:87, Src: TexasIns_00:0d:5e:d4:03
6LoWPAN
Internet Protocol Version 6, Src: fe80::212:4b00:d5e:d403 (fe80::212:4b00:d5e:d403), Dst: fe80::212:4b00:d6a:dc87 (fe80::212:4b00:d6a:dc87)
User Datagram Protocol, Src Port: 61617 (61617), Dst Port: 61618 (61618)
Data (19 bytes)
```

Figure 20. Two nodes are able to simultaneously send 19 data bytes every 100 milliseconds.

In this 6LoWPAN data transfer capability test I tried to evaluate ability of 6LoWPAN to handle data transfer when several nodes simultaneously start to transmit data frames. The reason for simultaneous data transfer test is quite obvious – synchronized nodes will acquire sensor’s values and send data to server in equal time point. The results show that it is possible to simultaneously send up to 18 data bytes every 50 milliseconds, however for higher data rates and time periodicity some improvements should be done. One of possible solutions is send data with random period, this will require to have data buffer on devices.

5.5 Summary

The results show that synchronization precision of both developed protocols is under 1 millisecond, or rather 600 microseconds in case of first protocol and 200 microseconds in case of second synchronization protocol (synchronization frequency ones in 30 seconds). To have constantly time difference between nodes and server less than 1 millisecond, synchronization should be initialized by server at least every 2 minutes in case of protocol version 1 and every 4 minutes in case of second protocol. Synchronization periodicity once in every 2-4 minutes will not over flood existing wireless sensor network. Time to synchronization seems very long, however I think it could be improved by adding random delay before sending unicast frame to server asking

for synchro. Another improvement in both protocols might be using more efficient compression, in my examples compression threshold was configured so that data less than 15 bytes will not be compressed, but when every node is asking server for synchro, they send unicast frame with data length of 2 bytes – this frame appears uncompressed, so the total UDP frame length is 74 bytes long. Unfortunately I don't have results with more efficient compression, because I found this Contiki OS specific when measurements were already done. Nevertheless I believe this could improve time to synchronization and 6LoWPAN data transfer capacity, but of course this does not influence precision of synchronization.

6 Conclusion and future improvements

During this thesis work I have studied the working principles of 6LoWPAN in IEEE 802.15.4 based networks. Knowledge of IPv6 also improved. Get familiar with Contiki-OS – operating system for IoT. Implemented UDP client-server communication and base on this solution, realized 2 modified versions of synchronization protocol based on IEEE 1588-2008 standard.

Deployed test environment and measurement results show that even with software realization of PTP the synchronization precision of sub-millisecond is achievable – 600 microseconds in case of first protocol and 200 microseconds in case of second synchronization protocol. To achieve such precisions, nodes should be synchronized at least every 30 seconds.

Data transfer capability tests show that two nodes could continuously and simultaneously transmit 18 data bytes every 50 milliseconds or 30 data bytes every one second. The data rates could be higher if nodes transmit packets in different time, with a gap for example one millisecond. Future work will be mainly based on improvements described above.

6.1 Future improvements

The first thing to improve in synchronization process is to take timestamps in MAC layer right after message received or just before sending out. This definitely will improve synchronization accuracy, otherwise generating timestamps at application level introduce error, since going up or down the protocol stack requires time which can't be determined and highly depends on workload of operating system or device in total. Generating timestamps at MAC layer, I think shouldn't be a very complex solution, since Contiki-OS is supplied with full available source code and it is easy to make changes and custom configurations.

The next step is to implement support of boundary clock and/or transparent clocks. 6LoWPAN based wireless sensor network with mesh topology could be deployed on a large area, where some end-devices are not directly connected to DAG root. If upstream nodes are simply ordinary clocks, not acting as boundary clocks, there is now way for downstream nodes to be synchronized with master clock. The solution for this issue seems

very plain – 6LoWPAN nodes already support route-over mode and storing mode is enabled, hence nodes hold routing and neighbour tables. So it seems logically to implement boundary clock on top of RPL. Of course this is only in case of software implementation of IEEE 1588-2008 standard, without specific hardware solution.

The last improvement I want to mention is to provide grand master with real time and date. In current tests grand master (6lowpan border router, or UDP server, or gateway) sends out its general purpose timer values, which is not time like UNIX time since epoch. Here any kind of solution could be implemented, since 6LBR is running on top of Raspbian in Raspberry Pi 3, which has Ethernet and WIFI interfaces, it uses mains power supply and it is more powerful comparing to normal microcontrollers.

Tests and measurement result showed that with implemented solution the achievable synchronization precision is in sub-millisecond range. With described improvements the accuracy could be increased, system become more stable and reliable, this gives opportunity for real field deployment and usage in wireless sensor networks.

References

[1] The Internet Engineering Task Force (IETF), “Transmission of IPv6 Packets over IEEE 802.15.4 Networks”, RFC 4944, 2007

[2] The Internet Engineering Task Force (IETF), “Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks”, RFC 6282, 2011

[3] The Internet Engineering Task Force (IETF), “Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)”, RFC 6775, 2012

[4] John C. Eidson, “Measurement, Control, and Communication Using IEEE 1588”, Springer, pp 3-7, 2006

[5] Zhao Dengchang, An Zhulin, and Xu Yongjun, “Time Synchronization in Wireless Sensor Networks Using Max and Average Consensus Protocol”, 2012 http://journals.sagepub.com/doi/pdf/10.1155/2013/192128, visit date: 7.05.2017

[6] The Internet Engineering Task Force (IETF), “Internet Protocol, Version 6 (IPv6) Specification”, RFC 2460, 1998

[7] International Organization for Standardization, “Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model”, ISO/IEC 7498-1:1994, 1994

[8] IANA, “IPv6 Special-Purpose Address Registry”, 2006, https://www.iana.org/assignments/iana-ipv6-special-registry/iana-ipv6-special-registry.xhtml, visit date: 7.05.2017

[9] The Internet Engineering Task Force (IETF), “IPv6 Stateless Address Autoconfiguration”, RFC 4862, 2007

[10] The Internet Engineering Task Force (IETF), “Neighbor Discovery for IP version 6 (IPv6)”, RFC 4861, 2007

[11] IEEE 802.15 WPAN Task Group 4 (TG4), “IEEE Standard for Low-Rate Wireless Networks”, IEEE Std 802.15.4-2015, 2015

[12] Internet Protocol for Smart Objects (IPSO) Alliance, “RPL: The IP routing protocol designed for low power and lossy networks”, 2011 http://www.ipso-alliance.org/wp-content/media/rpl.pdf, visit date: 7.05.2017

[13] Texas Instruments, “CC13x0, CC26x0 SimpleLink™ Wireless MCU Technical Reference Manual”, 2017 http://www.ti.com/lit/ug/swcu117g/swcu117g.pdf, visit date: 7.05.2017

[14] CETIC 6LBR, https://github.com/cetic/6lbr/wiki, visit date: 7.05.2017

[15] "Contiki: The Open Source OS for the Internet of Things", http://www.contiki-os.org/, visit date: 7.05.2017

[16] Texas Instruments, "Contiki-6LOWPAN", http://processors.wiki.ti.com/index.php/Contiki-6LOWPAN, visit date: 7.05.2017

[17] Adam Dunkels, "Protothreads", http://www.dunkels.com/adam/pt/, visit date: 07.05.2017

[18] Adam Dunkels, "The ContikiMAC Radio Duty Cycling Protocol", 2011 http://dunkels.com/adam/dunkels11contikimac.pdf, visit date: 7.05.2017

[19] IEEE Std, "1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", 2008

[20] Texas Instruments, "MSP-EXP430G2 LaunchPad Development Kit User's Guide", 2016

[21] Wireshark, https://www.wireshark.org/, visit date: 7.05.2017

[22] Live Traffic Capture and Sniffer for IEEE 802.15.4 networks, https://github.com/g-oikonomou/sensniff, visit date: 7.05.2017

[23] PuTTY: an SSH and telnet client, http://www.putty.org/, visit date: 7.05.2017

[24] Figure 1, "IPv6 header format", http://www.techietek.com/2014/11/13/hlim-icmpv6-packet/ visit date: 7.05.2017

[25] Figure 3, "IEEE 802.15.4 and IEEE 802.11n spectrum comparison", http://www.mdpi.com/2076-3417/5/4/1882/htm, visit date: 7.05.2017

[26] Figure 6, Figure 7, "Differences between mesh-under and route-over", "3 scenarios of IPv6 header compression", http://www.ti.com/lit/wp/swry013/swry013.pdf, pp 7, 9, visit date: 7.05.2017