Tallinna Ülikool

Digitehnoloogiate instituut

Andmevahetusprotokolli loomine Asjade Interneti kursuse jaoks

Seminaritöö

Autor: Hans Metsoja

Juhendaja: Jaagup Kippar

Autor:....., 2016

Juhendaja:....., 2016

Instituudi direktor:....., 2016

Autori deklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt

varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad,

kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

Kuupäev:

Allkiri:

Sisukord

Kasutatud mõisted

IOT – Internet of things ehk asjade internet, arusaam kus iga väike seade on ühendatud võrku ning suudab teiste seadmetega rääkida, edaspidi IOT

IPv4 – IP protokolli neljas versioon, edaspidi IPv4

IPv6 –IP protokolli kuues versioon, edaspidi IPv6

HMAC – (Hash-based message authentication code) on konstruktsioon leidmaks sõnumi autentimiskoodi mille komponendiks peale räsitavate andmete on salajane parool, edaspidi HMAC

M2M – machine-to-machine interface / protocol. Ehk side mis toimub ühest targast masinast teise, ning seal liikuv teave ei ole mõeldud inimesele kui lõppkasutajale

TLS - transpordikihi turbeprotokoll, edaspidi TLS

DTLS - kasutajadatagrammi protokolli turbeprotokoll, edaspidi DTLS

Sissejuhatus ning töö eesmärgid

Töö eesmärk on luua toetavaid õppematerjale Tallinna Ülikoolis loetavale asjade interneti kursusele.

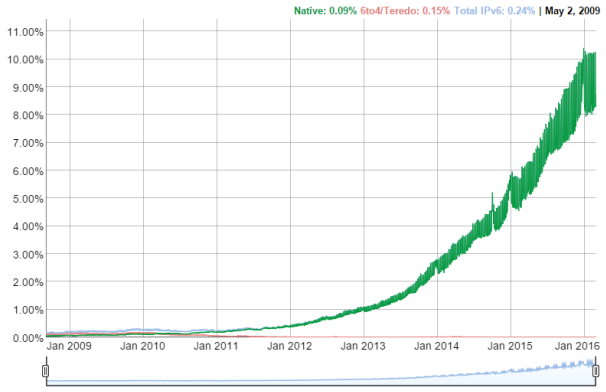
Asjade internet on suhteliselt uus valdkond ning eesti keelseid materjale väga ei eksisteeri, kuid kõik tahavad sellega tegeleda ning sellest kirjutada. Protokolle on kümnete kaupa ning ka seadmeid mida reklaamitakse kui Asjade Interneti seadmeid või platvorme on palju. Samuti puudub ühtne kokkulepe milliseid andmevahetus protokolle kasutada ning millistest standarditest kinni pidada. Kui välja jätta tööstuslikud seadmed kus on see kokku lepitud, siis kõik koduvidinad kipuvad olema erinevad ning neid tehakse kuidas juhtub.

Samuti on räägitud juba päris ammu visioonist, kus kõik seadmed on ühendatud internetti, kuid selle suureks piiranguks oli Ipv4 aadressruumi suurus, mis on 2^{32} (IETF RFC 791, 1981)

Samas ipV6 aadress-ruumi suuruseks on 2^{128} (IETF RFC 2460, 1998)

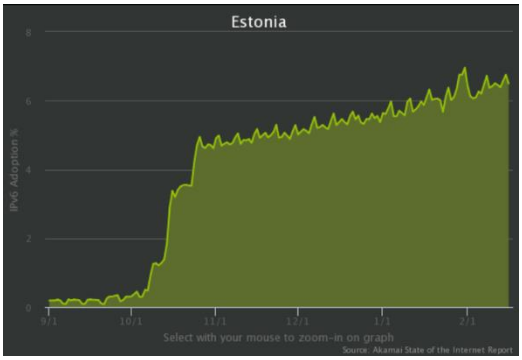
Kuigi esimene IP versioon 6 spetsifikatsioon ilmus aastal 1998 kulus üle 15 aasta, et disainiga jõutaks valmis ning interneti teenuse pakkujad hakkaks seda ka laiemale hulgale pakkuma.

Nagu näha jooniselt 1, on IPv6 hakanud laiemalt levima alles paari viimase aasta jooksul.



Joonis 1: Google IPv6 statistika

Samuti soosib kiire Ipv6 levik Eestis ka nii asjade interneti seadmete arendust ning ka laialdasemat kasutusele võttu(Tikan, 2015)(Joonis 2, IPv6 levik Eesti.



Joonis 2. IPv6 deployment in Estonia end of 2014 (source Akamai Internet Report)

Samuti on ka IpV6 oma eelkäijast turvalisem, sisaldades juba kohe IPSEC turva protokolli, mis IP v4 protokolli lisati hiljem juurde. (IPv6Now, kuupäev puudub)

Töö registreerimise said sõnastatud eesmärgid:

- Luua kursuse jaoks sissejuhatavat ning andmevahetusele keskenduvat õppematerjali
- Valida ning põhjendada seadmete valikut
- Luua väike näidisrakendus mitme erineva riistvara platvormi jaoks
- Keskenduda andmevahetus protokollidele näidis rakenduse alusel

1. Asjade interneti olemus

1.1. Mis on asjade internet

IOT ehk asjade internet on maailm, kus suhtlevad omavahel erinevad sensorid, automaatika seadmed ning teenused mis ei vaja ilmtingimata inimese sekkumist, kuigi võivad koguda informatsiooniga ka inimese kohta; üks selline näide on näiteks tervise andmeid koguvad käepaelad. Näide seadmest mis aga saab toimida autonoomselt on temperatuuri andur mille järgi tark kodu reguleerib kütet, või autode loendur ja ultraheli-kaugusandur mille järgi valgusfoori tsükleid reguleeritakse ning mis saadab ka infot selle kohta kesksele seadmele.

Samas aga ei loeta asjade interneti seadmeteks meil igapäevaselt kasutusel olevaid ja internetti ühendatud seadmeid nagu süle- ja tahvelarvutid, telekad, nutitelefonid, mängukonsoolid. (Duffy, 2014)

Enda olemuselt keskendub asjade internet kõige rohkem M2M sidele (Masinast masinasse)

1.2. Asjade interneti seadmetest üldiselt

Asjade interneti seadmete maailm on väga kirju ning erineb ka selletõttu, et iga üks defineerib asjade internetti natuke erinevalt. Samas võiksime jagada seadmed kolme suurde kategooriasse.

- Tööstuslikud Automaatika seadmed

  Siia alla võiks võtta targad voolumõõtjad, DIN liistudele kinnitatavad seadmed, andurid, releed. Näiteks kasvõi Droid4Control seadmed (http://www.droid4control.com)
  Antud seadmed on pigem mõeldud ehitajatele või arenduseks ning ei ole tingimata lõppkasutaja sõbralikud - lõppkasutaja ei hakka elektrikilpi 220 voldiga töötavad seadet ise paigaldama.

- Lõppkasutaja asjade interneti seadmed

  Siin all võime vaadata kõiki lõppkasutajale mõeldud seadmeid, mis teevad küll elu mugavamaks kuid on tihti suletud lähtekoodi ja protokolliga seadmeid mis töötavad vaid omaenda ökosüsteemis. Mainime küll need antud seminaritöös ära, kuid ei hakka neid rohkem lahkama. Siin võiks heade näidetena välja tuua Philips Hue targad elektripirnid (http://www2.meethue.com/en-xx/), pulssi ning kehalist aktiivsust jälgivad käepaelad ning ka Telia targa kodu lahenduse (https://www.telia.ee/era/muud-teenused/kodujuhtimine)

  Kahjuks aga uurimise ja õppimise mõttes pole siit palju peale ideede võtta.

- Asjade interneti arendusplatvormid

  Kõige huvitavam kategooria on aga asjade interneti arendusplatvormid, siia alla lähevad kõik mikrokontrollerid, sensorid ja arendusplaadid, mis on võimelised jooksutama enda kirjutatud programmikoodi, mille tarkvara ja mõnikord ka arhitektuur on täiesti avatud.

Samuti ei ole see jälle lõpptarbijast kodukasutajale õige valik, kui antud seminaritöö keskendub just neile õppe-eesmärkidel. Ning ka teadlikumal tehnikahuvilisel õnnestub just neid seadmeid kasutades endale ideaalne nutikodu luua.

Siin saab vaadelda kõiksugu mikrokontrollereid ning nende arendusplaate, näiteks Arduino, MSP430 LaunchPad, STM mikrokontrollerid ning PIC mikrokontrollerid. Väikesed, madala voolutarbega mikroarvutid mille külge ühendada sensoreid ja panna teistega suhtlema. Antud seadmete puhul võiks välja tuua aga ka mõne miinus punkti. Reeglina nõuab antud kontrolleritele arendamine C keele oskust, ning samuti on antud platvormide arvutusvõimsus ja mälu väga piiratud. (512kb – 2MB mälu) (Microchip(C), kuupäev puudub)

Samas tehakse ka võimsamaid väikearvuteid.

Kõige kuulsam on Raspberry Pi mikroarvuti millest on antud töö kirjutamise ajal juba välja tulnud kolmas põlvkond. Kuid on ka alternatiive nagu Banana Pi, Cubieboard ning Intel Galileo.
Sellised platvormid tarbivad küll rohkem voolu ja on suuremad, kuid on võimelised jooksutama kas windows, linux või android operatsioonisüsteemi, tehes neile arendamise tunduvalt lihtsamaks. Samas saab koodi kirjutada ja testida ka tavalise arvutiga kuna tarkvara on sama.

Antud tabel näitab, kui olulised on nende erinevat tüüpi mikroarvutite parameetrid. Aga õige tuleb valida loomulikult vastavalt kasutajaloole. Ainult temperatuuri lugemiseks ei ole otstarbekas valida suur ja suure voolutarbega protsessor.

Tabel 1 Erinevad asjade interneti seadmeplatvormid

Nimi	Intel Galileo	Rpi 3 põlvkond	Arduino Uno	PIC24F*
Protsessor	400MHz	1.2GHz x4 tuuma	16MHz	40MIPS
Mälu koodile	512 kb	SD kaart	32 kb	128KB
RAM	256MB	1GB	2 kb	16kb
Voolutarve	550mA	400mA	<50mA	<25mA
Suurus	13x7 cm	8,5 x 5,5 cm		
Hind	75$ ~ 70€	35€	20€	1€

1.3. Kursuse jaoks valitud seadmed ja tehnoloogiad

Antud seminaritöös osutusid valituks RaspberryPI mikroarvuti ning Arduino Uno mikrokontroller.(Tabel 2, valitud seadmed)

Tabel 2 Valitud asjade interneti seadmed

RaspberryPI mikroarvuti spetsifikatsioon:	Arduino Uno spetsifikatsioon:
SoC: Broadcom BCM2837 CPU: 4× ARM Cortex-A53, 1.2GHz GPU: Broadcom VideoCore IV RAM: 1GB LPDDR2 (900 MHz) Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy Storage: microSD GPIO: 40-pin header, populated Ports: HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)	Microcontroller ATmega328P Operating Voltage 5V Input Voltage (recommended) 7-12V Input Voltage (limit) 6-20V Digital I/O Pins14 (of which 6 provide PWM output) PWM Digital I/O Pins 6 Analog Input Pins 6 DC Current per I/O Pin20 mA DC Current for 3.3V Pin 50 mA Flash Memory 32 KB (ATmega328P) of which 0.5 KB used by bootloader SRAM 2 KB (ATmega328P) EEPROM 1 KB (ATmega328P) Clock Speed 16 MHz

Mõlemad seadmeid said valitud kahe põhilise omaduse tõttu.

Esimene neist on hind ja kättesaadavus. Tegemist on suhteliselt odavate seadmetega, mis pigem maanduvad hinnaskaala madalasse otsa või keskele. Ning osta saab neid Eestis mitmelt pakkujalt.

Teiseks otsustavaks punktiks sai just seadmete populaarsus. Kuigi eksisteerib alternatiive on Pi ja Arduino ühed populaarsemad seadmed, see tähendab, et abimaterjale ja manuaale leiab kergelt ja igal teemal. Samuti on olemas tehnilise toe foorumid kust alati abi küsida. Kuigi alternatiivsem lahendus võib olla natuke odavam, toodetakse nii Pi kui Arduino jaoks lisaseadmete ja andurite komplekte mis teevad eriti mugavaks nende platvormide kasutamise õppetöös.

Arduino arendus toimub Arduino enda arenduskeskkonnas C keele laadses programmeerimis keeles.

Raspberry Pi kasuks räägib ka Linuxi operatsioonisüsteem. Serveri rakenduse loomiseks kasutatakse Pythonit ning MySQLi. Kuna Python on arhitektuurist sõltumatu keel ning veebi- ja andmebaasiserveri saab paigaldada ka ARM arhitektuuriga protsessoriga arvutile, saab kasutada arendustööks ka puhtalt ainult arvutiklassi arvutit ning kui kood hakkab valmima saab selle kolida üle Raspberry Pi serverile.

Autor märgiks siinkohal ära, et olenevalt saada olevast arvutus võimsusest ning vaja minevast programmi efektiivsusest ja kiirusest ei pruugi Python olla alati kõige parem valik. Kuid hetkel teeb Pythoni kasutamine selle töö tunduvalt lihtsamaks ning programmi poolt tehtavate operatsioonide hulk on nii väike, et ka kiirus ei saa takistuseks. Samuti aitab kahe täiesti erineva seadme valik harjutada sama programmi realiseerimist erinevates keeltes erinevatele platvormidele.

2. Asjade interneti seadmetel kasutatavad protokollid

Enne kui lähme protokolli disainimise juurde teeme lühikese tutvustuse protokollidest mis on juba olemas ja erinevatel asjade interneti lahendustel kasutusel.

2.1. HTTP/REST

Kõige klassikalisem interneti protokoll, loodud dokumentide vahetuseks. RESTful programmiliides on üle HTTP toimiv protokolli täiendus mis teeb eriti lihtsaks ja inimloetavaks API enda.

Positiivse poole pealt on selline protokoll kindlasti kõige laiem ning universaalsem. Serverile ei ole vaja arendada mõnda muud teist protokolli rääkivat liidest juurde vaid saab kasutada juba olemas olevat.

Miinustest võiks mainida, et antud protokoll on päris mahukas ning HTTP päised võivad võtta päris palju ruumi andme edastusel. Ning piiratud mäluga kontrolleritel tuleb ka kogu HTTP protokolli tugi seadmele sisse programmeerida.

Siin toome võrdluseks näiteks välja kui palju erinevad moodsate asjade interneti seadmete riistvaralised parameetrid, nutikellade näitel:

Tabel 3 Nutikellade võrdlus

	Pebble Time (2015)	Apple Watch (2015)	Samsung Galaxy Gear
Protsessor	100MHz	520MHz	1 GHz kahe tuumaline
Muutmälu	128kb	512MB	512 MB
Salvestusmälu	16MB	8GB	4GB

Nagu tabelist näha, siis mõnele seadmele ei tekitaks HTTP/REST päringud üldse probleemi kuid väiksema mäluga seadmete korral kus on erilisem operatsioonisüsteem tuleb aga teistsugust lahendust kasutada.

2.2. LLAP

LLAP (ingl. K. lightweight local automation protocol) on kerge kaaluga andmeside protokoll mis loodud just väikestele mikrokontrolleritele. Iga sõnum on 12 tähemärgi pikkune ja inimloetav.

Sõnum on formaadis aXXDDDDDDDD, kus a näitab alati sõnumi algust, XX ütleb mis seadmega on tegemist ning D on andmed. Kui andmeid on vähem, vooderdatakse(ingl. K padding) sõnum – märkidega täis pikkuseni.

Selline protokoll töötab väga hästi täiesti isoleeritud ning suletud võrgus, kus andmete kohale jõudmine ei pea olema garanteeritud. Turvalisust ei ole tagatud aga võimalus seda teha oleks üle sokli ning TLS-i . Samuti puudub sõnumil kontrollsumma millega õigsuses veenduda ning vahetatavate sõnumite pikkusel on oluline piirang.

2.3. XMPP

XMPP on päris vana protokoll, mis pärineb aastast 1999 ning formaliseeriti aastal 2004 (IETF RFC 3921, 2004)

Originaalselt mõeldud olekuinfo ja kiirsõnumite edastamiseks on. XMPP puudused on samuti sisse ehitatud turva, vajadusel tuleb täiendavalt sisu krüpteerida ning võtmevahetus välja mõelda. XMPP sõnumi sisu on XML sarnane, ning seetõttu on ka paketi saatmise üldkulu(ingl. K overhead) päris suur.

Positiivse poole pealt on XMPP serverit lihtne käima saada, on ühilduv ning teistele teada olev protokoll ning enamvähem inimloetav.

2.4. CoAP

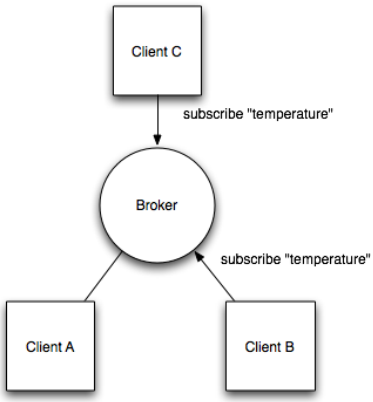
CoAP on sarnaselt HTTP-le dokumendi transpordi protokoll, erinevusega olles disainitud piiratud mäluga seadmetele (Jaffey 2014). Coap transpordikiht on UDP. Sisse on ehitatud väga algeline teenuse kvaliteedi kontroll, kus saab küsita kinnituspakette ning turvalisuse tagamiseks on võimalik kasutada DTLS-i või siis krüpteerida paketi sisu eraldi.

CoAp sensorid on tavaliselt ise serveri rollis, ning nende külge tuleb ühendada et lugeda sensori väärtust või lülitada mõnda funktsiooni.
Reeglina mõeldud lahendustesse kus seade peab rääkima vaid ühe teise seadmega (näiteks kesk seade)

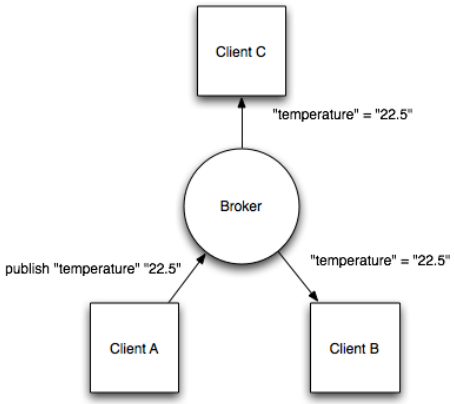
2.5. MQTT

Tegemist on vaba standardiga, mis mõeldud just M2M side jaoks ning välja arendatud IBM´i poolt. Töötab klient-server mudelil IP protokolli peal, iga sõnum on iseseisev.

Iga seade peab registreerima endale huvi pakkuva teema kesk-serveris, ning seejärel saab saata sellel teemal kõigile uuendusi. Mõeldud kasutamiseks juhtudel kus üks seade peab saatma enda sõnumi paljudele.(Joonised 3 ja 4)



Joonis 3 MQTT and CoAP, IoT Protocols, 2014



Joonis 4MQTT and CoAP, IoT Protocols, 2014

MQTT-l on sisse ehitatud teenusekvaliteedi kontroll (QOS). MQTT toetab kasutajanime ja parooliga autentimist ning kogu side saab üle TLS-i käima panna.

Samas tahab MQTT ikkagi TCP tuge ning teemade nimed võivad olla väga pikad mis muudab selle protokolli ebapraktiliseks Zigbee ja teistes sarnastes võrkudes (Jaffey, 2014)

3 Näidis protokolli disain

3.1 Sissejuhatus protokolli disaini

Kasutajaloona mõtlesin välja maakoha temperatuuri andmete kogumise.

Täismahus lahendus mida looma ei hakata võiks koosneda kesksest serverist, Raspberry PI baasil, mis kogub andmed kohalikku andmebaasi ning on võimeline üle 3G modemi ka neid välismaailma saata, ning kohalikust raadiovõrgust, mille külge ühendavad end Arduino baasil tehtud sensorid temperatuuri ning niiskuse lugemiseks. Antud seminaritöö raamides jäätakse raadioliides välja ning luuakse see lahendus kasutades tavalist Ethernet-tüüpi kaabliga ühendust. Samuti teeb see lihtsamaks ka praktikumis võrguliikluse pealt kuulamise. Seadmeid simuleeritakse tavalise võrku ühendatud Arduinoga, kuigi päris rakenduses peaks ka sensori panema aku toitele, tekitama patarei tühjenemise hoiatuse või päikese paneeliga laadimise

Ka ei minda antud seminaritöös lõplikult valmis lahenduseni välja vaid disainitakse protokolli raamistiku, mida iga huviline saab täiendada vastavalt oma soovile.

3.2 Legend

Protokolli disain on asi, mida reeglina ei tehta ise, nagu ka krüptograafia lahenduste disainimine.

Autor teadvustab endale, et alati on lihtsam võtta mõni valmis protokoll mille jaoks on olemas teegid, mida on testitud ning kõige tähtsamana – mis on teiste seadmetega juba ühilduv. Ise disainimine mitte õppe-otstarbel on tavaliselt ajamahukas ning kulukas ja arusaadav vaid juhtudel kus tõesti on seadmed väga spetsiifilised või suurte riistvaraliste piirangutega.

Olenevalt kasutusest peaks alustama mingisugusest stsenaariumist, mida üritatakse saavutada, mille jaoks on seda antud protokolli vaja. Mis seadmetel see peab jooksma ning mis on transpordi kiht.

Sellest tulenevalt saab paika panna nii turva kui ka funktsionaalsed nõuded. Samuti võiks protokolli teha võimalikult paindliku ja pigem disainida raamistikuna, et hilisem kasutamine mujal oleks lihtsam ja laiendavatus suurem. Näitena, et koodi mitte sisse kirjutada LAMP, 0xFA, vaid hoopis [seadme ID]=0x0013a 0xFA annab võimaluse defineerida „0x0013a” mistahes seadmeks.

3.3 Loodud protokoll ning mida silmas pidada

Loodud näidis-protokoll on olemuselt äärmiselt lihtne.

Kasutusel on TCP-sokkel ning protokolli koodil endal on vaid 3 sisendit.(Tabel 3)

Tabel 4 Loodud asjade interneti protokolli väljad

Klient_ID	Väli on mõeldud seadme identifitseerimiseks ning eri seadmetest tulevate päringute eristamiseks. Samuti saab registrit seadme GUID´ide järgi hallata näiteks dünaamiliselt läbi veebiliidese ning määrata neile seadme tüüpe külge (valgus, temperatuur jne)
even_ID	Even_ID väli on mingi käsu või sündmuse väljendamiseks kasutusel. Näiteks saab sellega öelda kas tegemist on valguse või temperatuuri andmetega Set_temp, set_light Või hoopis aja küsimisega Get_time
request	Määramatu väli mis on päringu enda sisu, siin võib olla mingi numbriline väärtus määramaks temperatuuri, valgus tugevust või ka defineeritud sõna. Samuti võib see olla ka hoopis null väärtusega ja ebaoluline kui even_ID on näiteks „ALERT” ning sellisel puhul on vaja käivitada häire

Pakett ise koosneb viiest väljast, kus juurde pannakse ka automaatne ajatempel süsteemi kella järgi ning sõnumi autentsuse kood(Tabel 4)

Tabel 5 Loodud asjade interneti protokolli pakett

Klient_id (seadme idendifikaator)	Even_id (sündmuse tüüp)	Ajatempel	Request(sisu)	Autentsuse kood

Autentsuse kood arvutatakse räsi funktsiooniga üle nelja eelmise välja, kasutades ka salajast võtit.

3.4 Loodud protokolli turva ning mida silmas pidada

Loodud protokolli turbe mehhanismiks on ainsana kasutusel HMAC-räsi sõnumist. Kuna legendi järgi on tegemist sensoritelt andmete lugemise protokolliga (valgus, liikumine, õhuniiskus, temperatuur)

Peab autor siinkohal oluliseks andmete õigsust. HMAC-räsi kaudu saame kontrollida ning veenduda, et keegi ei oleks neid andmeid muutnud. Samuti kuna kuulaval serveril ei ole parooli ja kasutajanime saab igaüks kes on samas võrgus sinna andmeid saata, HMAC-räsi võrdlemine võimaldab aga loobuda valede pakettide töötlemisest. Hajutatud teenusetõkestamise ründe vastu see muidugi ei aita, kuid sellist tüüpi ründega saab süsteemi üle koormata ka lihtsalt katkisi pakette saates.

Kuna temperatuur kuuris ei pruugi olla väga salajane info ei ole rakendatud ka transpordikihi turvet. Kõik info mis üle võrgu liigub on paketinuhutajale (Ingl. K. Packet Sniffer) loetav. See teeb ka rakenduse kasutamise arvutiklassides lihtsamaks – kõike mis liigub võrgus saab lihtsalt pealt kuulata silumise eesmärgil.

Rakenduse turvalisus seisneb HMAC-räsis.

HMAC-räsi tööpõhimõte on äärmiselt lihtne. Võetakse sõnumist räsi, kuid üheks osaks on ka salajas hoitud võti (IETF RFC 2104)
Kui teine pool saab üle võrgu sama sõnumi mille lõpus on HMAC-räsi, on võimalik võtta sisendiks sõnumi enda sisu, ning teades salajast parooli arvutada uuesti üle HMAC-rasi ja nende kahe võrdlusel on võimalik tuvastada kas sõnumit on muudetud. Samuti ei tea seda salajast parooli ka kolmas osapool, kelle saadetud sõnumite HMAC-räsi ei vasta kindlasti oodatule.

Kriitiline komponent antud protokollis on ka ajatempel. Esiteks oleks vaja teada, mis ajal millist sensori väärtust raporteeriti. Teiseks tagab ajatempel selle, et HMAC-räsi on erinev ka siis kui sama sensor saadab mitu korda järjest sama numbrilist väärtust.

Räsi funktsioonina on kasutusel SHA-256, mis RIA tellitud krüptograafiliste algoritmide elutsükli uuringu kohaselt turvaline (Willemson, 2013)

3.5 Serveri rakendus ja test programm

Server on loodud Python programmeerimis keeles, eelkõige seetõttu, et seda oleks kiire ning lihtne läbi mängida, ning samas saab arendada ja testida enda igapäeva arvutis ning valmis koodi liigutada üle Pi serveri peale.

Juba Pythoniga kaasas olevatest teekidest on kasutusel:

```
import hashlib
import hmac
import socket
import time
```

Samas ei oska Python väga hästi SQLi külge ennast ühendada, selle lahendamiseks peame kasutama MySQL connectorit. Seminaritöö praktilise osa kirjutamise ajal oli viimane Pythoni versioon 3.4 ning sellele vastav MySQL connector ka olemas, kui töö enda kirjutamise ajal on Pythonist väljas versioon 3.5 kuid MySQL connector on endiselt versioon 3.4jale (https://dev.mysql.com/downloads/connector/python/)

Selletõttu soovitame endiselt kasutada ka Pythoni natuke varasemat versiooni

```
import mysql.connector
```

```
secretKey = 'j344rfdas'
```

määrab ära kasutusel oleva saladuse.

SQL kood on kirjutatud kujul, kus serveri mitte töötamine ei mõjuta sensoritest lugemist. Kuigi antud juhul sellisel rakendusel praktilist väärtust ei oleks. Küll aga saab õppetöös siiski näha liikuvaid pakette.

```
try:
    cnx = mysql.connector.connect(user='iot', password='userpar1',
                                  host='127.0.0.1',
                                  database='iot')
    print("SQL connected")
except:
    print("SQL error")
```

Paketi loomise kood on alljärgnev. Siit on näha ka kõigi väljade kokku võtmine ning järgnevalt ka räsi arvutamine üle kõigi väljade.

```
def create_packet(klient_id, even_id, request):
    timest = (int(time.time()))
    packet = "{}.{}.{}.{}.{}".format(klient_id, even_id, timest, request, bytes.decode
(hashpacket("{}.{}.{}.{}".format(klient_id, even_id, timest, request))))
    return packet
```

```
def hashpacket(packet):
    bytes = str.encode(packet)
    key = str.encode(secretKey)
    hashedpacket  = ( hmac.HMAC(key, bytes,hashlib.sha256))
    return str.encode(hashedpacket.hexdigest())
```

Kood jookseb koheselt ka Pi serveri peal, kuid meeles tuleb pidada, et lisaks Pythonile tuleb ka paigaldada SQL server ja luua andmetabel õige struktuuriga. Ning RaspberryPI-l puudub sisemine patarei kella hoidmiseks, seega tuleb kell seadistada õigeks peale igat voolukatkestust. Õnneks saab operatsioonisüsteemi seadistada kasutama ajaserverit, mistõttu paneb operatsioonisüsteemi kell end õigeks kohe kui seade saab toimiva võrguühenduse.

Lisaks on ka väike jupp test-koodi, et protokolli ja serveri rakenduse saaks lihtsamalt läbi mängida ilma Arduino sensorita ja veenduda, et see töötaks.

Väljavõte Pythonis realiseeritud test-kliendist:

```
def create_packet(klient_id, even_id, request):
    timest = (int(time.time()))
    packet = "{}.{}.{}.{}.{}".format(klient_id, even_id, timest, request, bytes.decode
(hashpacket("{}.{}.{}.{}".format(klient_id, even_id, timest, request))))
    global requ
    requ = "{}.{}.{}.{}".format(klient_id, even_id, timest, request)
    return packet

def saada(pakett):
    try:
        s.connect((host, port))
        print("Success connecting to ")
        print(host," on port: ",str(port))
        data = str.encode(pakett)
        print ("sending :", pakett)
        s.sendall(str.encode(pakett))
        tagasi = (s.recv(2048))
        print (tagasi)
    except socket.error as e:
        print("Cannot connect to ")
        print(host," on port: ",str(port))
        print(e)

alfa = (create_packet(123,4,"Close"))
saada(alfa)
```

3.6 Arduino sensori rakendus

Arduinole koodi kirjutamine ja testimine on juba natuke keerulisem. Esiteks on keel erinev.

Teiseks on koheselt platvormist tingitud erinevused. Arduinole tuleb kogu pakettside kood sisse kompileerida kuna operatsioonisüsteemi kui sellist Arduinol ei ole. Teiseks puudub ka seadmel kell. Kuna kellaaega võetakse tarkvaraliselt ning antud seade istub võrgus, on mõistlik seadme käivitudes lugeda ajaserverist õige kell.

Teekidena on kasutusel

```
#include <sha256.h>
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
#include <Time.h>
```

Kõige probleemsemaks osutus Arduino Krüptograafia teek (Arduino Cryptosuite) , mis ei tahtnud väga hästi kõige uuemas arenduskeskkonnas tööle hakata. Õnneks leidus antud teegist ka kohandatud versioon mis ühildub uuema Arduino keskkonnaga. (CryptoC)

Kuna antud töö käsitleb protokolli osa, sai sensorite väärtused simuleeritud:

```
void loe_valgus()
{
  int a = random(3,15);
  rand_valgus = String(a * 100);
  Serial.println(rand_valgus);
}
```

Alljärgnev osa näitab paketi loomise realisatsiooni ning signeerimist Arduino keeles:

```
void create_packet(String event, String request)
{
     pakett = device_ID + "." + event + "." + now() + "." + request;
}
```

```
void create_hmac (String hmac_data)
{
  Sha256.initHmac((uint8_t*)"j344rfdas",9);
  Sha256.print(hmac_data);
  printHash(Sha256.resultHmac());

}
```

```
void printHash(uint8_t* hash) {
  int i;
  for (i=0; i<32; i++) {
    temp = temp + ("0123456789abcdef"[hash[i]>>4]);
    temp = temp + ("0123456789abcdef"[hash[i]&0xf]);
  }
}
```

```
void compile_packet(String data_in, String hmac_in)
{
      payload = data_in + "." + hmac_in;
}
```

Võrgusokli ning ajaserveri koodi osas toetuti suuresti arduino kodulehel olevatele näidetele.

3.7 Parandused ja tegemata tööd

Nagu näha siis antud puhul ei ole tegemist terviliku rakenduse või lahendusega, vaid materjaliga läbi mille õppida ja õpetada protokolli loomist. Antud lähtekoodi saab võtta aluseks kui raamistiku mis juba töötab Raspberry Pi ja Arduino peal. Autor on jätnud lahtiseks väga palju, lootes, et ehk keegi saab seda teemat edasi arendada ning antud protokolli täiendada, turvalisemaks muuta.

Samas peaks antud töö andma baas teadmised tudengile mille peale ehitada üles oma asjade interneti rakendus. Kuna nii seadmeID/tüüp ning käsu ID/tüüp on jäetud defineerimata saab laiendada antud tööd veel kõvasti.

Antud protokoll on mõeldud lahtisena ning seda saab võrguliiklust jälgides pealt kuulata, seega seda ei tooks puudusena välja. Küll aga võiks mainida, et kõige rohkem tähelepanu peaks pöörama sisendi ja väljundi valideerimisele (kas on õige pikkusega, kas on lubatud sümbolid)

Pythoni puhul on seda lihtsam teha ja vigast sõnumit saab ignoreerida, kuid C keele puhul on liiga suur sisu puhvri ületäituminse põhjuseks(Owasp), mis lisab aga turvariske.

Samuti on antud kood hetkel ühesuunaline, antud tööd tasuks laiendada ja luua ka kahesuunaline sideprotokoll millega saaks saata sõnumi kättesaamise kinnitusi ning ka kellaaega või muid käsklusi sensoritele.

Samas võimaldab antud lahendus juba saata aku tühjenemise teadet sensorist kesk seadmele. Veel tuli piiranguna välja Arduino Uno enda mälumaht. Kuna kasutusel olevad teegid on mahukad õnnestus ära kasutada üle poole saada olevast mälust. Kui nüüd seda laiendada veel juhtmevabaks sideks vajalike teekidega ning sensorite lugemisega võib juhtuda, et Arduino Uno mälumahust jääb väheks.

Kokkuvõte

Antud töö pidi andma kerge ülevaate ja sissejuhatuse asjade interneti seadmetesse, protokollidesse ning tegema ka sissejuhatuse protokolli disaini. Töö tulemusena valmis paar rida Pythoni näidiskoodi serveri realiseerimiseks ning ka Arduino-kliendi kood.

Autor tahaks mainida, et antud tööl on väärtus vaid õppe-eesmärkidel ning kui soov on luua lõppkasutajale mõeldud rakendus tasuks pigem uurida, millist juba olemas olevatest asjade interneti protokollidest kasutada. Kuid kui on soov ise õppida ja harjutada peaks töö raames tehtud näidiskood andma selleks juba toimiva aluse mida vastavalt vajadusele täiendada.

Samuti õppis autor, et seadmeid tuleb hoolikamalt valida ning valik paremini läbi mõelda. Oli puhtalt juhus, et Arduino Unost piisas. Kuigi kood ei olnud mahukas, võtsid enamus ruumi ära vajalikud teegid.

2014/2015 õppeaastal viidi ka antud töö raames läbi 2 loengut ning üks praktikum, millest esimene loeng keskendus sissejuhatusse, üld infosse, IPv6 olemusse ning levikusse ja seadmetele üldiselt, vaadati, katsuti ka erinevaid näiteid asjade interneti seadmetest

Teine loeng keskendus protokolli disaini, võrgu liikluse ning transpordi meediumile. Korrati üle ka krüptograafia põhialused ning räägiti mida silmas pidada. Praktikumi osas testiti ja prooviti oma koodi näite varal kirjutada, ning seda täiendada mõneks valmis rakenduseks töö autori kaasabil

Kasutatud kirjandus.

1. Tikan Tarko, 2015, IPv6 development in Estonia, viimati loetud 2016. Märts, https://labs.ripe.net/Members/tarko_tikan/ipv6-deployment-in-estonia
2. RFC 791, 1981, Internet Protocol, viimati loetud Märts 2016 aadressil, https://tools.ietf.org/html/rfc791
3. RFC 2460, 1998, Internet Procotol version 6, viimati loetud Märts 2016 aadressil, https://tools.ietf.org/html/rfc2460
4. Google IPv& statistika, viimati loetud Märts 2016 aadressil, https://www.google.com/intl/en/ipv6/statistics.html
5. Duffy, Jim, (2014), *8 things that are not IOT, viimati loetud Märts 2016*, aadressil, http://www.networkworld.com/article/2378581/internet-of-things/8-internet-things-that-are-not-iot.html
6. *Intel Galileo datasheet* (2013), viimati loetud märts 2016 aadressil, http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-g2-datasheet.html
7. Microchip *PIC datasheet,* viimati loetud Märts 2016 aadressil, http://www.microchip.com/design-centers/16-bit
8. Arduino, *Arduino Uno datasheet*, viimati loetud 2016 aadressil, https://www.arduino.cc/en/Main/ArduinoBoardUno
9. Raspberry Pi, *andmeleht*, viimati loetud Märts 2016 aadressil, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
10. IBM, (2010) *MQTT protokoll*, viimati loetud Märts 2016, aadressil, http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html
11. Jaffey Toby, kuupäev pole teada*, MQTT ja CoAP protokollide võrdlus*, viimati loetud Märts 2016, aadressil, https://eclipse.org/community/eclipse_newsletter/2014/february/article2.php
12. IETF, (2013), *Constrained Application Protocol (CoAP),* viimati loetud Märts 2016 aadressil, https://tools.ietf.org/html/draft-ietf-core-coap-18
13. Willemson, Jan (2013), *Krüptograafiliste algoritmide elutsükli uuring II*, viimati loetid märts 2016, aadressil, https://www.ria.ee/public/PKI/kruptograafiliste_algoritmide_elutsukli_uuring_II.pdf
14. Arduino, *UDPNTP client*, viimati loetud Märts 2016 aadressil, https://www.arduino.cc/en/Tutorial/UdpNtpClient
15. Arduino, *Chat client*, viimati loetud Märts 2016 aadressil, https://www.arduino.cc/en/Tutorial/ChatClient
16. Cathedrow, 2010, *Arduino Cryptosuite*, viimati loetud aadressil https://github.com/Cathedrow/Cryptosuite
17. Snowda, 2014, *Arduino CryptoC*, viimati loetud aadressil https://github.com/Snowda/CryptoC