

UNIVERSITETI I PRISHTINËS “HASAN PRISHTINA”

FAKULTETI I INXHINIERISË ELEKTRIKE DHE KOMPJUTERIKE

AUTOMATIKË E KOMPJUTERIZUAR DHE ROBOTIKË



PUNIM DIPLOME

Tema:

Rregullimi digjital me komunikim nëpërmjet cloud

Mentori:

Prof. Ass. Dr. Lavdim Kurtaj

Kandidati:

Enes Hoxha

Prishtinë, Janar 2021

Abstrakt

Për të krijuar pajisje gjithmonë aktive dhe gjithmonë të lidhura na nevojitet që fillimisht të krijojmë shërbime gjithmonë aktive dhe gjithmonë te lidhura. Nga një artikull i Salesforce në Forbes thotë se nga viti 2020 do të jenë më shumë se 75 miliard pajisje të lidhura në internet, të njohura si pajisjet IoT. A ju u ka ndodhur që të mendoni dy here nëse keni ndal te gjitha dritat në shtëpi? Besoj qe po! Dijgitalizimi po ndodhë dhe atë shumë shpejtë. Sot nuk nuk po flitet se si te kontrollohen pajisjet nga dora e njeriut, por se si ato të komunikojnë dhe rregullojnë njëra-tjetren edhe nëse nuk janë të lidhura fizikisht apo pranë njëra-tjetrës.

Për të ofruar komunikim në mes të pajisjeve me mikrokontroller nëpërmjet cloud pa vonesa të mëdha kohore ($<=100\text{ms}$) kanë zënë përdorim të lartë platformat e shpërndara për transmetim. Kyqja e rrjetit 5G do të ju mundësoj pajisjeve që të komunikojnë me njëra-tjetren pa vonesa të mëdha kohore si në rastin e veturave vet-vozitëse e cila në komunikacion do të mund të komunikoj me cdo pajisje që ndodhet pranë, si telefonat, orat e mençura të kembesorve, me semaforët e shtrirë në qytete. Vetura do të mund të marr informata dhe ti procesoj ato me ndihmën e Cloud. Realizimi i këtij eksperimenti në këtë temë ka për qellim njohjen sadopak të këtyre rregullimeve digitale me komunikim nëpërmjet cloud, duke përdorur sisteme ekzistuese si cloud dhe platformat e shpërndara të transmetimit. Rregullimi me komunikim i cili shihet në këtë temë nuk është rregullim në kohë reale por afér kohës reale, pasi qe këto sisteme bëjnë delegimin e informatave të transmetuara. Problemi i cili është analizuar jep kuptim veprimeve asinkrone dhe ndërprerëse, një komandë mund të jetë veprim ndërprerës me një pajisje tjetër të mençur, si në rastin e veturës vet-vozitese e cila ndalon ne momentin kur një njeri kalon rrugën apo në semafor paraqitet drita e kuqe. Eksperimenti tregon që rregullimi digjital me komunikim nëpërmjet cloud është i realizueshëm dhe vonesat kohore janë të arsyeshme brenda vlerave të lejueshme.

Abstract

To create always on and always connected devices, first we must create always on and always connected services. From an article of Salesforce on Forbs it says that from year 2020 there will be more than 75 billion devices connected to internet known as IoT Devices. Have you ever thought twice about turning off all the lights in your house? I believe so! Digitization is happening and amazingly fast. Today we are now talking about how to control devices from our hands, but how devices can communicate and fix each-other, even if they are not physically connected or near each-other.

To provide communication between devices with microcontrollers via cloud without major time delays ($\leq 100\text{ms}$) have been widely used distributed streaming platforms. The arrival of 5G network will enable devices to communicate with each other without major time delays as in the case of self-driving vehicles which in traffic will be able to communicate with all nearby devices such as smartphones, smart watches. Realization of this experiment on this theses aims somehow recognition of these adjustments with digital communication via cloud, using existing systems such as cloud computing and distributed streaming platforms. Adjusting the communication it seems in this theses is not real-time adjustment but it is near real-time, as these systems delegate the transmitted information. The problem that has been analyzed gives meaning to asynchronous and interrupting actions, a command can be an interrupt action to another smart device, as in the case of self-driving cars which stops when a person crosses the road or if the traffic light is red. The experiment shows that the digital adjustment with cloud communication is feasible and delays are reasonable within the allowable values.

Përbajtja

1 Hyrja	1
1.1 Hyrje	1
1.2 Motivimi	1
1.3 Përshkrimi i problemit	2
2 Teknologjitet e përdorura	4
2.1 Raspberry Pi 3	4
2.1.1 Specifikat teknike	5
2.1.2 GPIO	6
2.1.3 Antenat	7
2.2 Windows 10 IoT Core	7
2.3 Microsoft Azure	7
2.4 Andy X	8
2.5 Microsoft Visual Studio	9
2.6 GitHub	10
3 Lidhja dhe konfigurimi	11
3.1 Lidhja e Raspberry Pi 3	11
3.2 Instalimi i Windows 10 IoT Core	11
3.3 Konfigurimi i Andy X	13
4 Softueri, logjika e mikrokontrollerit dhe kompjajlimi	15
4.1 Ndarja e softuerit	15
4.1.1 Aplikacioni	15
4.1.2 Modelet e të dhënave	18
4.1.3 Logjika	19
4.2 Lidhja e MCU-ve dhe testimi i aplikacionit	26
4.2.1 Testimi i aplikacionit	27
4.3 Aplikacioni kompjuterik për lexim të të dhënave gjatë transmetimit	29
5 Analiza e eksperimenteve dhe rezultatet dalëse	31
5.1 Matja e vonesës kohore	31
5.2 Rezultatet nga matja e vonesës kohore në internet stabil	32
5.2.1 Vonesa kohore në 100 matje	32
5.2.2 Vonesa kohore në 200 matje	33
5.2.3 Vonesa kohore në 300 matje	34
5.2.4 Vonesa kohore në 400 matje	34
5.2.5 Vonesa kohore në 500 matje	35
5.3 Rezultatet nga matja e vonesës kohore në internet jo-stabil	36
5.3.1 Vonesa kohore në 100 matje	36
5.3.2 Vonesa kohore ne 200 matje	37
5.4 Rezultatet nga eksperimenti Test në mes të mikrokontrollerëve	38
6 Diskutime dhe konkluzione	40
7 Bibliografia	41

1 Hyrja

1.1 Hyrje

Komunikimi i pajisjeve digitale në mes vete dhe dirigjimi i tyre nëpërmjet internetit është shumë e kohës, me avancimin e sistemeve komunikuese është diçka e realizueshme lehtë dhe e sigurtë. Sot ekzistojnë shumë sisteme dhe standarde nëpërmjet të cilave mund të realizohen këto llojë komunikime. Përdorimi i platformave Cloud është duke u rritur dita ditës njëkohësisht dhe numri i pajisjeve qe përdorim çdo ditë është duke u rritur. Sot nëpërmjet telefonit mund të kontrollojmë dhe dirigojmë pajisjet që kemi ne shtëpi si ngrohësit e ujit, nxemjet qendrore, sistemin e ujitjes etj. Keto pajisje janë të lidhura në internet. Në këtë temë do të analizojmë komunikimin në mes dy mikrokontrollerve nëpërmjet internetit. Me daljen e gjeneratës së pestë të rrjetit celular 5G këto komunikime në mes pajisjeve do të lehtësohen ku gjithashtu do të rritet dhe përdorimi i tyre. Ky rrjet do të mundësoj realizimin e pajisjeve gjithmonë të ndezur dhe gjithmone te lidhur (*en. Always on and always connected devices*). Sot ekzistojnë shumë sensor të ndryshëm si ata të temperaturës, lagështis së ajrit, rrjedhjeve të ndryshme, presionit nëpër fabrika të ndryshme të cilat dërgojnë të dhëna në kohe reale në qendrat e kontrollit nëpërmjet platformave Cloud. Këto platforma kanë fuqi jashtëzakonisht të madhe mbi ato rezultate dhe historikut të tyre mund të aplikohen algoritme të ndryshme nëpërmjet të cilave mund të parashikohen edhe prishjet e pajisjeve. Në këtë mënyrë mund të zgjedhjet edhe humbjet që mund t'i kenë kompanitë në rastet e prishjeve të mëdha, një shembull të tillë mund të kemi tek termocentralet si ato që gjenden në Obiliq të Kosoves (prishjet e gjeneratorëve).

Pra, si eksperiment në këtë temë do të kemi komunikimin në mes dy mikrokontrollerëve nëpërmjet cloud dhe do të analizohen vonesat kohore dhe humbjet e informacionit. Komunikimi i këtyre dy mikrokontrollerve është realizuar mbi platformat e shpërndara për transmetim (*en. Distributed Streaming Platforms*).

1.2 Motivimi

Nga një artikull i **Salesforce** në Forbes thotë se nga viti 2020 do të jenë më shumë se 75 miliard pajisje të lidhura në internet.^[4] Ky shkrim ka qenë një motiv për realizimin e komunikimit në mes dy mikrokontrollerëve nëpërmjet internetit. Këta lloj sensor të cilët janë të lidhur në internet ndryshtë quhen edhe **Interneti i gjërave** (*en. IoT, Internet of Things*). IoT përfaqëson një fushe të re për të aplikuar dhe zhvilluar kërkime në teknologji. Interneti i gjërave varet nga ne! Sensorët dhe pajisjet e fiksuar në vetura, telefona, orë dore, markete, shtëpi, rrugë, ura, pajisje industriale, bujqësore dhe teknologjia e bartshme (*en. wearable*) të cilat krijojnë infomacion të ri nga i cili, ne sot kemi ndryshuar mënyrën e prodhimit, konsumit dhe mënyrën e jetesës. Këto pajisje IoT kanë hapur një horizont të ri mbi marrjen dhe analizën e informacionit. Aplikimet më të mëdha të pajisjeve IoT janë monitorimi në ndërmarrje dhe infrastruktura civile, sistemet e energjisë, kujdesit shëndetësor, argumented reality etj.

Të gjitha këto përdorime të pajisjeve IoT, sfida kryesore është komunikimi. Kjo është arsyja kryesore përse në këtë temë po e shtjellojmë problemin e komunikimit në mes dy mikrokontrollerëve.

1.3 Përshkrimi i problemit

Krijimi dhe realizimi i një shtëpie të mençur (*en. Smart Home*), lidhja fizike e të gjithë sensorëve dhe pajisjeve është shumë e shtrenjtë dhe thuajse e pamundur. Që të lidhim të gjithë mikrokontrolleret mes vete, këta mikrokontroller mund të kenë performanca dhe sisteme të ndryshme, prandaj lidhja e të gjithë mikrokontrollerve që janë në një shtëpi të mençur është shumë e vështirë dhe e shtrenjtë. Si shkak i këtyre problemeve janë paraqitur edhe standarde të ndryshme të komunikimit në mes pajisjeve të mençura (pajisjet që kanë MCU brenda) disa nga to janë, Wifi, ZigBee, Z-Wave, Bluetooth, Cloud (Internet).

Një nga zgjidhjet është që të ofrojmë një infrastruktur komunikuese dinamike në mes MCU-ve është krijimi i një sistemi të përbashkët i cili ka për detyrë që të lejoj dhe menaxhoj komunikimin në mes pajisjeve, këto sisteme sot ndodhen në platformat **Cloud**.

Pra, në momentin që kemi dërguar informacionin nga pajisja në Cloud ne kemi mundësi të krijojmë logjika të ndryshme. *Shembull: nëse kemi dy MCU, MCU01 ka një sensor për matjen e temperaturës, ndërsa MCU02 është një kondicioner, të dy kta MCU janë të lidhur në Internet. Logjika e punës së kësaj lidhje mund të realizohet në shërbimet që kemi në cloud i cili merr informacionin nga MCU, e përpunon atë dhe dërgon komandën në MCU02.*

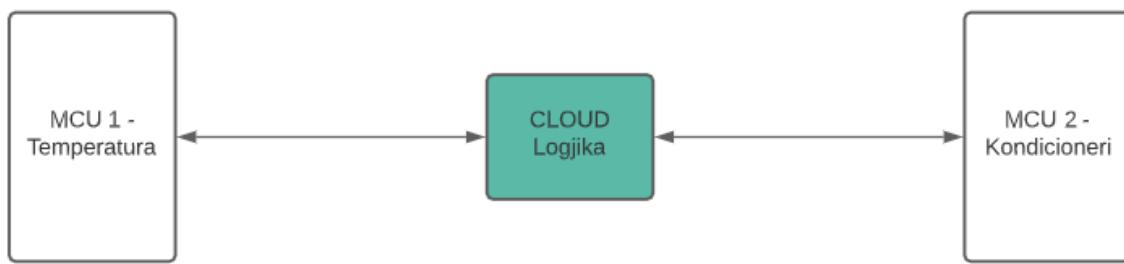


Figura 1: Reprezantimi grafik i shembullit

Nga ky shembull shohim që kemi mundësi të ndryshme të dizajnjimit dhe zhvillimit të IoT pajisjeve.

Nje ndër problemet më të mëdha është krijimi i pajisjeve që janë gjithmonë aktive dhe të lidhura (*en. Always On and always connected devices*), një gjë e tillë nuk është e mundur me përdorimin e Ueb shërbimeve (*en. Web-services*). Për të krijuar pajisje gjithmonë aktive dhe të lidhura duhet që të kemi shërbime gjithmonë aktive dhe të lidhura (*en. Always on and always connected services*). Këto shërbime duhet që t'i ngjajnë punës së trurit të njeriut i cili është gjithnjë aktiv dhe punon në logjik asinkrone (*en. asynchronous*). Logjika asinkrone e sistemeve është e lidhur ngushtë me arkitekturat asinkrone nëpërmjet të cilave mundësohen shërbimet

gjithmonë aktive. Për të mundësuar shndërrimin e shërbimeve në shërbime gjithmonë aktive dhe të lidhura ekzistojnë platformat e shpërndara të transmetimit (*en. Distributed streaming platforms*). Pra, për zgjidhjen e këtij problemi në këtë temë kemi përdorur **Andy X**, ky produkt është me kod burimor të hapur dhe mundeson veprimin asinkron në mes shërbimeve në rastin tonë MCU-ve, i cili bënë delegimin e kërkesës (*en. request*) dhe kthimin e një ngjarje (*en. event*)

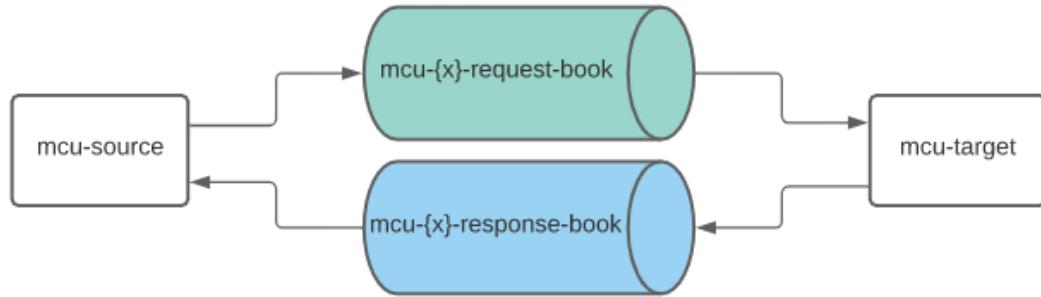


Figura 2: Reprezantimi grafik i lidhjës së dy mikrokontrollérve me Andy X

Njëra nga problemet tjera që mund të shfaqet gjatë komunikimeve nëpërmjet cloud-it dhe tek platformat e shpërndara për transmetim janë **vonesat kohore**, këtë problem do të trajtojmë në këtë temë dhe humbjet e informatës. Për komunikim në mes MCU-ve dhe shërbimit që ndodhë në cloud përdoret TCP (Transmission Control Protocol). TCP është protokoll komunikimi në internet i cili është connection-oriented dhe kontrollon lidhjen në mes klientit (MCU) dhe serverit (Cloud shërbimit) para se të dërgoj te dhënat.

Një tjetër problem që duhet diskutuar është edhe kostoja e realizimit të këtyre sistemeve duke e ditur që platformat Cloud nuk janë të lira per rastet kur kemi volumë të lartë të krijimit të informatave dhe transmetimit të tyre.

Në figurën në vazhdim shihet diagrami i lidhjes së dy mcu-ve nëpërmjet cloud me teknologjitet e përdorura të cilat do të analizohen në kapitullin në vazhdim.

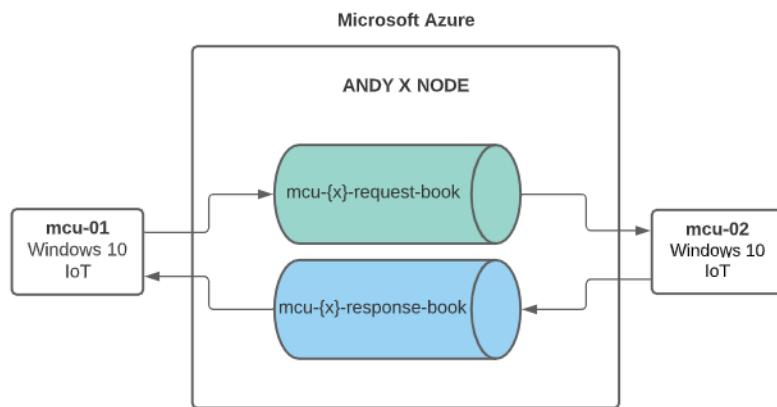


Figura 3: Reprezantimi grafik i rregullimit digital me komunikim nëpërmjet cloud

2 Teknologjite e përdorura

Në ditët e sotme ekzistojne shumë vegla të bazuara në teknologji të ndryshme të cilat mundësojnë zhvillimin e aplikacioneve të ndryshe me zbatime të ndryshme në jetën reale. Përveç ndryshimit në teknologji, ato ndryshojnë edhe në aspektin e përdorimit dhe mundësive që ofrojnë. Veglat e përdorura për realizimin e kësaj teme janë:

Dy **Raspberry Pi** mikrokontroller te cilëve gjate këtij dokumenti do t'iu referohemi si **MCU**, sistemi operativ me të cilin do të punojnë këta dy MCU është **Windows 10 IoT Core**. Platforma Cloud e cila është përdorur është **Microsoft Azure**, platforma e shpërndarë për transmetim e cila është përdorur është **Buildersoft Andy X**, si IDE është përdorur **Microsoft Visual Studio**, gjuha programuese e përdorur është **C#**, dhe vegla për të menaxhuar versionimin e kodit është **GitHub**.

2.1 Raspberry Pi 3

Raspberry Pi është një pllakë e vetme kompjuteri (*en. single-board computer*) e zhvilluar në Angli nga Fondacioni Raspberry Pi. Raspberry Pi në fillim si projekt është përdorur për të promovuar mësimdhënjen për shkencat kompjuterike dhe në vendet në zhvillim. Hardueri i kësaj pllakë ka evoluar në disa versione të ndryshme me karakteristika të ndryshme si tipi i procesorit, kapaciteti i memorik, përkrahja e rrjetit dhe lejimi i pajisjeve periferike si maus apo tastjer. Në bllok diagramin e dhënë më poshtë është prezantuar modeli B dhe B+, modeli A dhe A+, të cilat janë të ngjashme.

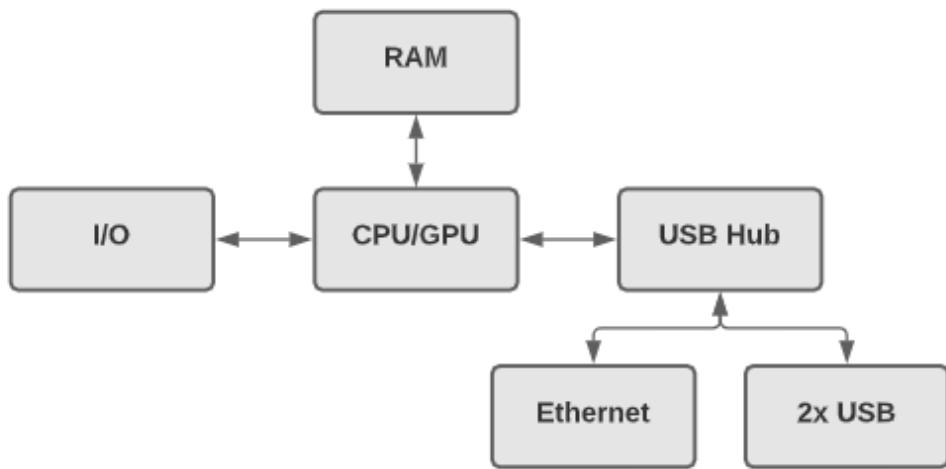


Figura 5: Diagrami i komponentëve të një Raspberry Pi 3

Sistemet operative me të cilat mund të punoj një Raspberry Pi janë : FreeBSD, Linux, NetBSD, OpenBSD, Plan 9, RISC OS, Windows 10 ARM64 dhe Windows 10 IoT Core.

2.1.1 Specifikat teknike

Procesori i vendosur në këtë mikrokontroller eshte **Broadcom BCM2387** chipset me performanca 1.2GHz me katër bërtthama me arkitekturë ARM Cortex-A52 (64bit) [1]



Figura 4: Raspberry Pi 3

Wireless LAN dhe Bluetooth 4.1

- IEEE 802.11 b/g/n Wi-Fi. Protokolet WEP, WPA, WPA2, algoritmet AES-CCMP (256bits) dhe shtrirje maksimal është 100 metra.
- IEEE 802.15 Bluetooth, Algoritmi i enkriptimit simetrik, Advanced Encryption Standard (AES) me qelës 128-bit, dhe shtrirje maksimal është 50 metra.

GPU

- Dual Core Video Core IV® Multimedia Co-Processor. Perkrah Open GL ES 2.0, me përshtypjetues harduerik (*en. hardware-accelerated*) Open VG, and 1080p30 H.264 high-profile dekoder.

Memorja

- 1GB LPDDR2

Sistemi operativ

- Ky mikrokontroller mund të punoj me versione të ndryshme të Linux apo Windows 10 IoT të cilat startojnë nga një SD kartë të vogël.

Dimensionet

- 85 x 56 x 17 milimetra

Fuqia

- Micro USB me input 5V, 2.5A

Lidhjet:

Interneti

- 10/100 BaseT Ethernet Socket

Dalja e videos

- HDMI (rev 1.3 dhe 1.4)
- Composite RCA (PAL dhe NTSC)

Dalja e audios

- Dalja analoge 3.5mm,
- HDMI,
- USB 4 x USB 2.0 Lidhesit

GPIO Lidhesit

- 40 pin 2.45mm (100mil) me zgjerim të kokës: 2x20 pika
- Oferon 27 GPIO pina si dhe +3.3V, +5V dhe GND linja të furnizimit

Lidhesi i kamerës

- 15-pin MIPI Camera Serial Interface (CSI-2)

Lidhesi per Ekran

- Display Serial Interface (DSI)

Vend per Memory Card

- Shty/tërheq Micro SDIO

2.1.2 GPIO

Raspberry Pi 3 karakterizohet me 40-pina për qëllime gjeneralë input-output (GPIO). Cdo pajisje ekzistuese e lidhur me GPIO do të funksionoj pa modifikim ndryshimi i vetëm është një ndërprerës në të cilin UART është i eksposuar në kunjat GPIO, por e cila trajtohet brenda nga sistemi operativ. Ne figurën në vazhdim i kemi të paraqitura 40 GPIO kunjat që ndodhen në Raspberry Pi 3.

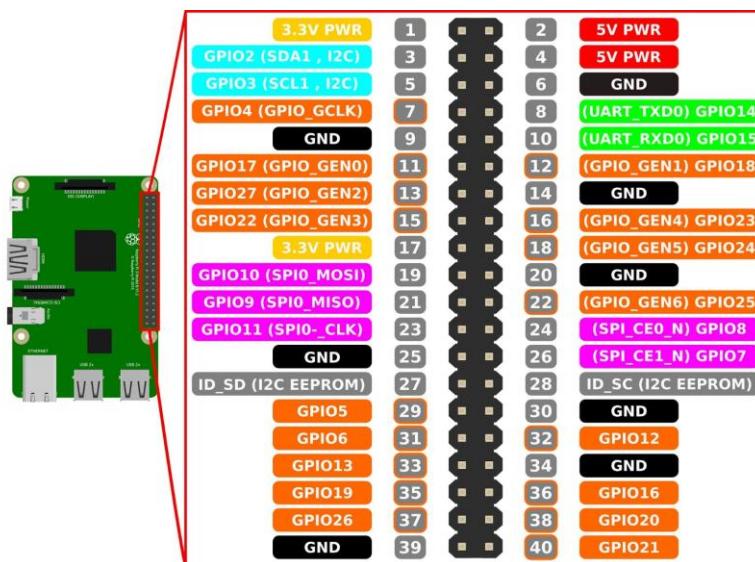


Figura 4.1: GPIO portat që ndodhen në Raspberry Pi 3 [8]

2.1.3 Antenat

Pasi që Raspberry Pi 3 i ka antenat e implementuar në pllakë nuk është e nevojshme për të lidhur antena të jashtme. Për të mbajtur madhësinë e mikrokontrollerit sa më të vogël këto radio antena nuk janë shumë të mëdha por janë mëse të mjaftueshme që të marrin sinjal të Wireless Lan dhe Bluetooth edhe përtej mureve.

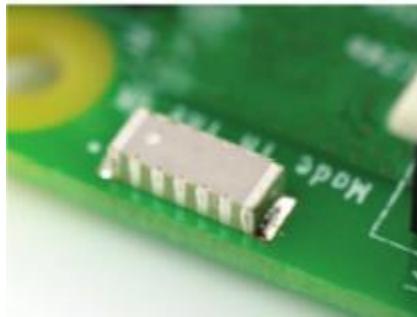


Figura 4.2: Radio Antenat e implementuara në Raspberry Pi 3

2.2 Windows 10 IoT Core

Windows 10 IoT Core është një version i Microsoft Windows 10 i cili është i optimizuar për pajisje të vogla me apo pa ekran.



Figura 6: Windows 10 IoT Logo

Microsoft Azure apo shpesh e referuar si Azure, është shërbim për cloud computing e krijuar nga Microsoft për të krijuar, testuar, shpërndarë dhe menaxhuar aplikacionet dhe shërbimet e krijuara nëpërmjet qendrave të të dhënave (Data Centers). [11]

2.3 Microsoft Azure

Microsoft Azure apo shpesh e referuar si Azure, eshte shërbim per cloud computing e krijuar nga Microsoft per te krijuar, testuar, shprendar dhe menaxhuar aplikacionet dhe shërbimet e krijuara nëpërmjet qendrave te te dhënave (Data Centers). [3]



Figura 7: Microsoft Azure Logo

Azure ofron softuer, platformë dhe infrastrukturë si shërbim tek klientët e saj së bashku me shumë gjuhë të ndryshme programuese, vegla dhe frameworks nga vetë Microsoft dhe nga palët e treta (third-party) sisteme. Azure gjithashtu ofron në shkallë të gjerë virtualizim në qendrat e te dhënave te Microsoft në të gjithë botën me më shumë se 600 shërbime të ndryshme. Ne jemi duke e përdorur vetëm një shërbim të Azure i cil quhet App Service, në të cilin e kemi të hostuar Andy X.

2.4 Andy X

Buildersoft Andy X është një platformë me kod burimor të hapur i cili është dizajnuar për të ofruar shkallë të lartë të performancës për transmetim të të dhënave në mes sistemeve të ndryshme duke lejuar arkitektura asinkrone ku shërbimet janë gjithnjë aktive dhe të lidhura. Kjo platformë gjithashtu ruan të gjitha të dhënat që kalojnë nëpër të me një shërbim tjetër që njihet si **Andy X Data Storage**. Kjo platformë në kohën e krijimit të kësaj teme ofron një pako-mjetesh (toolkit) vetëm per C# .NET. Të dhënat në këtë sistem janë të izoluara nëpër libra (*en. Books*)

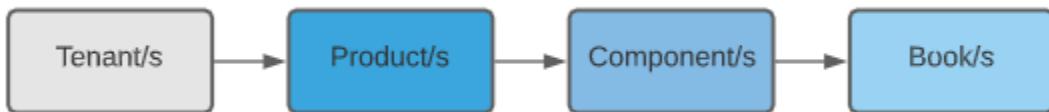


Figura 8: Struktura organizative e te dhënave brenda Andy X

Të dhënat që transmetohen në Andy X njihen si **Mesazhe**, të cilat mund të përmbajnë struktura të ndryshme të informacionit. Në figurën 8 shihet struktura organizative për ndarjen e rrjetit të shënimeve. Një tenant mund të ketë shumë produkte, ku secili produkt mund të ketë komponentë brenda dhe secila komponentë ka libra të ndryshme të cilat përmbajnë mesazhet e dërguara nga burimi i informatës dhe pranuara nga destinacioni, ku një mesazh mund të pranohet (lexohet) nga shumë shërbime (mikrokontroller) njëkohësisht.

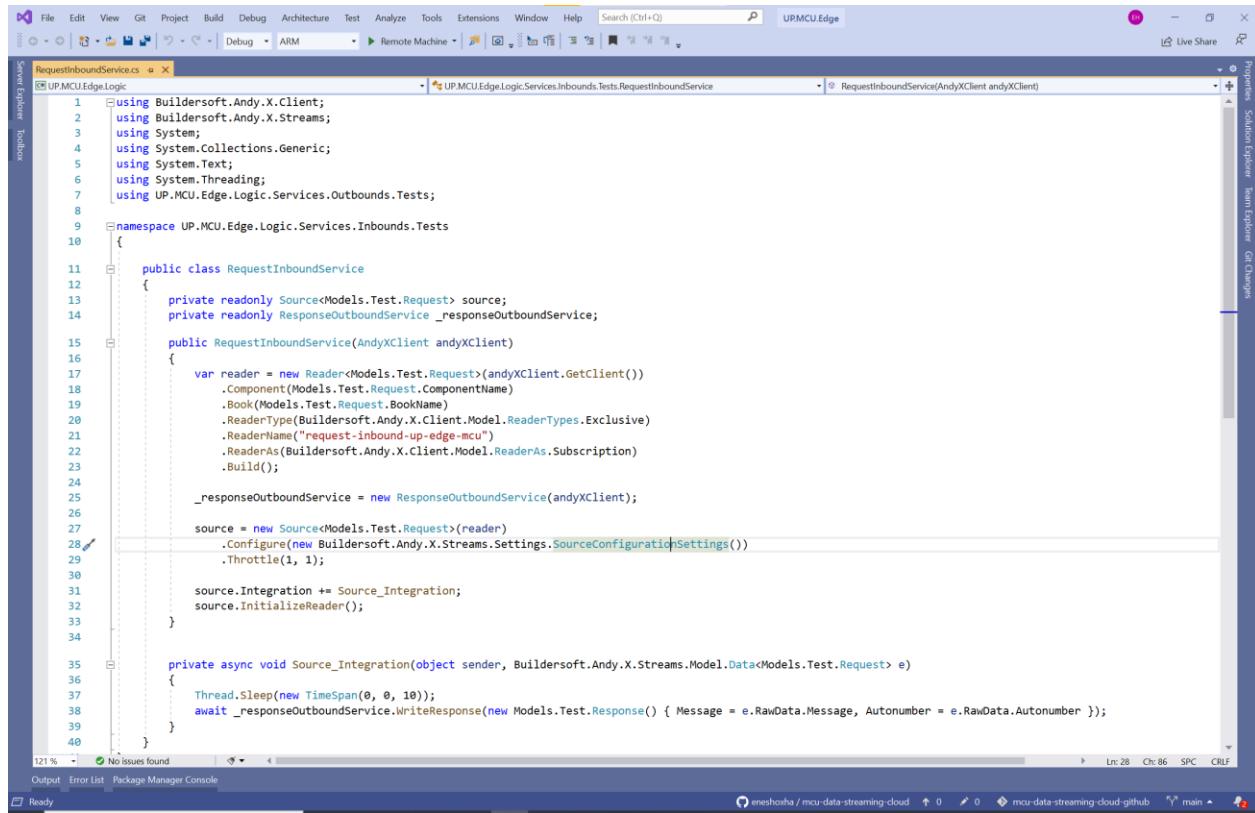
Tipet e përdoruesve janë **Shkrues** (*en. Writer*) dhe **Lexues** (*en. Reader*). Këto dy tipe janë pjesë të biznes logjikes brenda MCU-se dhe jashtë tij. Nje MCU mund të jetë njëkohësisht Lexues dhe Shkrues. MCU është Lexues atëherë kur pranon një kërkesë nga një MCU tjetër. Një MCU është shkrues atëherë kur bënë kërkesë për veprim me një MCU tjetër, ose kthen mbrapa një ngjarje/rezultat (*en. event/result*). Në këtë rast dy MCU-të mund të janë Master apo Slave. Programeri i cili duhet të zhvillojë logjikën e MCU-se nuk i intereson për transmetimin e mesazhit në destinacion se a është dërguar me sukses apo jo, pasi që këtë proces e garantojnë platformat e shpërndara për transmetim. [10]



Figura 8: Andy X Logo

2.5 Microsoft Visual Studio

Microsoft Visual Studio është një ambient i integruar për zhvillim(Integrated development environment IDE) nga Microsoft. Ky softuer përdoret për të zhvilluar programe kompjuterike si Uebfaqe, aplikacione të uebit, shërbime të uebit dhe aplikacione mobile në ekosistemin Windows si Windows API, Windows Forms, Windows Presentation Foundation(WPF),Windows Store Apps dhe UWP (Universal Windows Platform). Në ditët e sotme kjo IDE përdoret më tej Windows-it siç është rasti me .NET Core i cili është framework Cross Platform (aplikacionet zhvillohen për sisteme të ndryshme si Linux, Android, iOS). Visual Studio përkrah rreth 36 gjuhë të ndryshme programuese si C, C++, Visual Basic, C#, F#, JavaScript, TypeScript, XML, HTML, CSS, Python, Ruby, Node.js, Java. [5]



The screenshot shows the Microsoft Visual Studio interface with the following details:

- Title Bar:** UPMCU Edge
- Code Editor:** RequestInboundService.cs (UP MCU.Edge.Logic.Services.Inbounds.Tests.RequestInboundService)
- Code Content:**

```

1  using Buildersoft.Andy.X.Client;
2  using Buildersoft.Andy.X.Streams;
3  using System;
4  using System.Collections.Generic;
5  using System.Text;
6  using System.Threading;
7  using UP.MCU.Edge.Logic.Services.Outbounds.Tests;
8
9  namespace UP.MCU.Edge.Logic.Services.Inbounds.Tests
10 {
11     public class RequestInboundService
12     {
13         private readonly Source<Models.Test.Request> source;
14         private readonly ResponseOutboundService _responseOutboundService;
15
16         public RequestInboundService(AndyXClient andyXClient)
17         {
18             var reader = new Reader<Models.Test.Request>(andyXClient.GetClient())
19                 .Component(Models.Test.Request.ComponentName)
20                 .Book(Models.Test.Request.BookName)
21                 .ReaderType(Buildersoft.Andy.X.Client.Model.ReaderTypes.Exclusive)
22                 .ReaderName("request-inbound-up-edge-mcu")
23                 .ReadersAs(Buildersoft.Andy.X.Client.Model.ReaderAs.Subscription)
24                 .Build();
25
26             _responseOutboundService = new ResponseOutboundService(andyXClient);
27
28             source = new Source<Models.Test.Request>(reader)
29                 .Configure(new Buildersoft.Andy.X.Streams.Settings.SourceConfigurationSettings())
30                 .Throttle(1, 1);
31
32             source.Integration += Source_Integration;
33             source.InitializeReader();
34
35         }
36
37         private async void Source_Integration(object sender, Buildersoft.Andy.X.Streams.Model.Data<Models.Test.Request> e)
38         {
39             Thread.Sleep(new TimeSpan(0, 0, 10));
40             await _responseOutboundService.WriteResponse(new Models.Test.Response() { Message = e.RawData.Message, Autonumber = e.RawData.Autonumber });
41         }
42     }
43 }

```
- Status Bar:** 121 %, No issues found, Output, Error List, Package Manager Console, Ready.
- Bottom Status:** eneshoxha / mcu-data-streaming-cloud, 0, 0, mcu-data-streaming-cloud-github, main, 9%, F11

Figura 9: Pamje e hapësirës zhvilluese në Visual Studio

Për dizajnimin dhe zhvillimin e logjikes në **MCU** kemi përdorur gjuhën programuese **C#** e cila është pjesë e **.NET Framework** zhvilluar nga **Microsoft**.

C# është gjuhë programuese me qëllime të përgjithshme në të cilën përfshihen static-typing, strong-typing, programim funksional, orientuar-në-objekte (class-based) dhe është e orientuar në komponentë. [2]

2.6 GitHub

GitHub është platformë zhvillimi e inspiruar nga mënyra se si kodohet një aplikacion, duke ofruar mbështetje për përdorim të hapur apo open source dhe për biznese. Kjo platformë mundëson publikim, shpërndarje dhe rishikim të kodit, menaxhim të projekteve dhe krijim të softuerëve së bashku me miliona zhvillues të tjerë [6]. Sistemi që përdor kjo platformë për realizimin e këtyre e këtyre funksioneve, quhet Git dhe është një sistem kontroll i verzionit i cili mundëson gjetjen e ndryshimeve në fajlla kompjuterik dhe i koordinon punët në këto fajlla mes shumë njerëzve. Pra, ai sistem përdoret për menaxhimin e kodit burimor në zhvillimet e softuerit [7]. Përdorimi i i GitHub e bën më të lehtë bashkëpunimin me koleg tjerë dhe ofron shikimin e verzioneve të mëparshme të punës. Njëra nga arsyet kryesore të përdorimit të kësaj platforme është mundësia e shqyrtimit të kodit nga komuniteti. Kritikat nga pjesa e komunitetit janë gjithmonë të mirë se ardhura, pasi që ndikojnë drejtëpërdrejtë në vlerësimin dhe përhapjen e aplikaciont. Vegla e ndërtuar në këtë projekt është e publikuar në sistemin GitHub, dhe aty mundësohet rishikimi i punës që është bërë në këtë projekt, dhe gjithashtu edhe vazhdimi i saj për ofrimin e karakteristikave të reja.

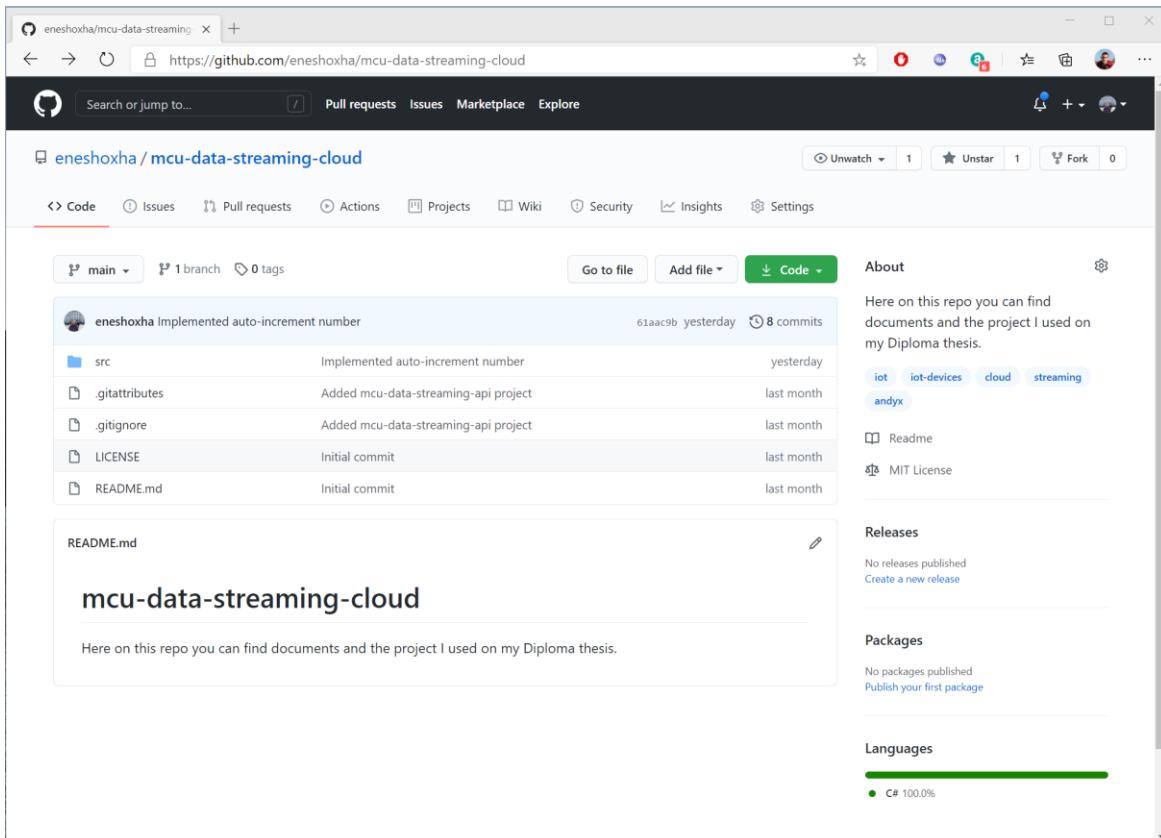


Figura 10: Pamje nga projekti i publikuar në GitHub [15]

3 Lidhja dhe konfigurimi

Në këtë kapitull do të analizojmë lidhjen e MCU-ve në internet dhe instalimin e Windows 10 IoT Core në dy MCU-te, gjithashtu do të bëjmë konfigurimin e Andy X për të lejuar komunikimin në mes dy MCU-ve.

3.1 Lidhja e Raspberry Pi 3

Fillimisht sigurohemi që i kemi të gjitha elementet e duhura haruderike si 2 Raspberry Pi mikrokontroller, një ose dy **UTP** kabllo për lidhje me internet, 2 kabllo **micro-usb** me output 5 volt të cilat nevojiten për lidhjen me rrymë të mikrokontrollerëve dhe dy sd-kartela mikro me kapacitet minimal 8GB të cilat do të nevojiten për të mbajtur sistemin operativ.

3.2 Instalimi i Windows 10 IoT Core

Për të instaluar **Windows 10 IoT Core** për Raspberry Pi mikrokontrollerët fillimisht nevojiten disa hapa.

- Të shkarkohet aplikacioni **Windows 10 IoT Core Dashboard**,
- Të lidhet SD-kartela me kompjuter,
- Të startohet aplikacioni **Windows 10 IoT Core Dashboard** si në imazhin në vazhdim

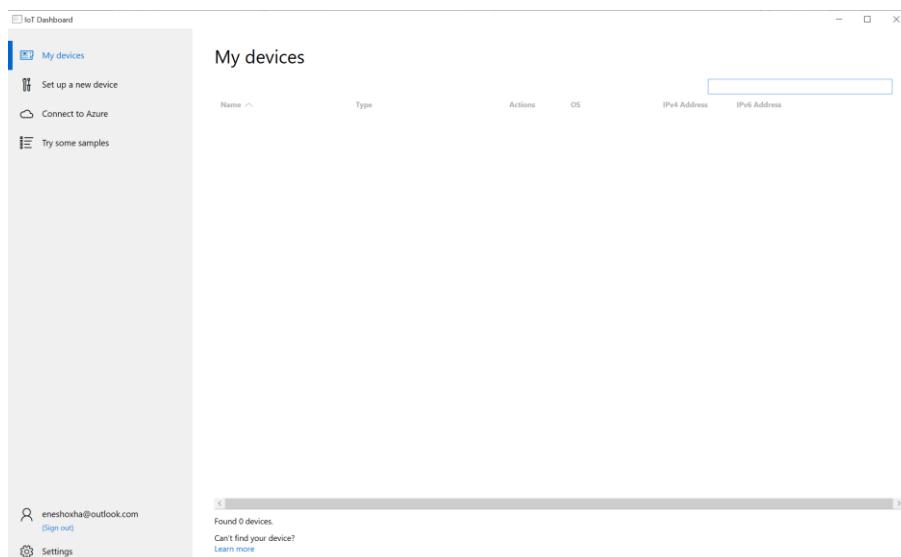


Figura 11: Windows 10 IoT Core Dashboard – My Devices

- Të hapet menya e dytë **Set up a new device**, si në imazhin në vazhdim

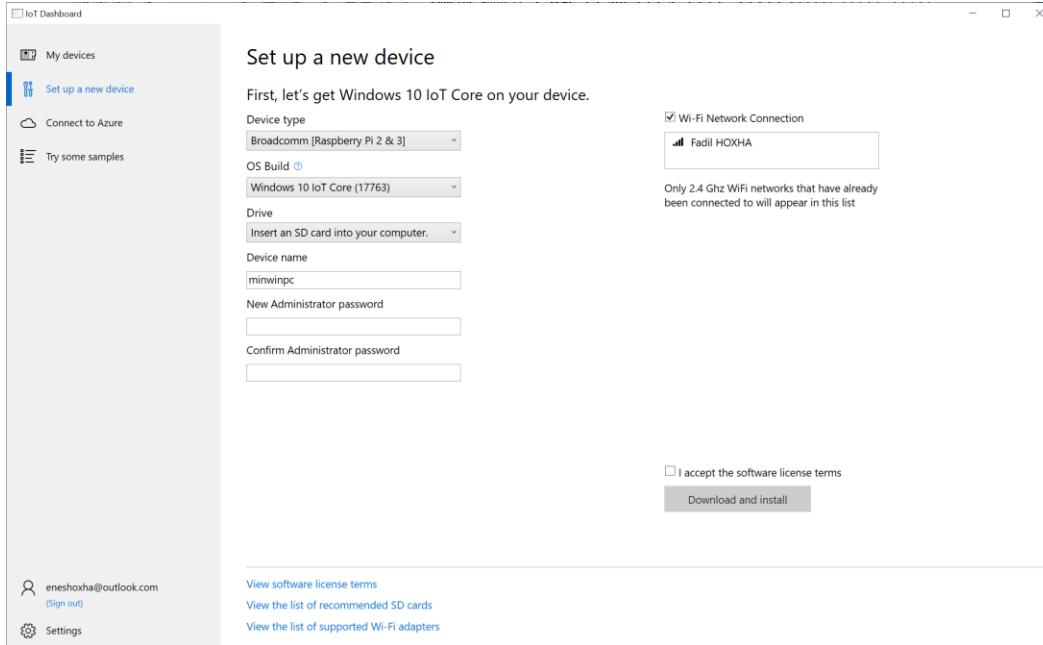


Figura 12: Windows 10 IoT Core Dashboard – Set up a new device

- Të zjidhet SD kartela tek fusha **Device**, të caktohet emri i pajisjes në rastin që e kemi ne për temë janë **bs-andy-mcu-01** dhe **bs-andy-mcu-02**, gjithashtu duhet të jepet edhe një fjalëkalim ku në rastin tonë e kemi **1234** për të dy mikrokontrollerët.
- Të pranohet licenca e sistemit dhe të klikohet butoni **Download and install**.
- Pasi që të bëhet instalimi për MCU-në e parë duhet të vazhdohen hapat e njejtë për MCU-në e dytë.
- SD-kartelat duhet të vendosen nëpër mikrokontroller dhe duhet që të ndezim dhe lidhim me rrjetë.
- Nëse njëri nga mikrokontrollerët do të lidhet me **HDMI** me një ekran të jashtëm do të paraqitet pas 5 deri 10 minutave (hera e parë) do shfaqet imazhi si në vazhdim



Figura 12: Windows 10 IoT Core in Raspberry Pi 3

Pasi që në ekran të kemi të paraqitur imazhin si në figuren 12, atëherë jemi gati për të vazhduar më tutje konfigurimin e parametrave të nevojshëm për Andy X.

3.3 Konfigurimi i Andy X

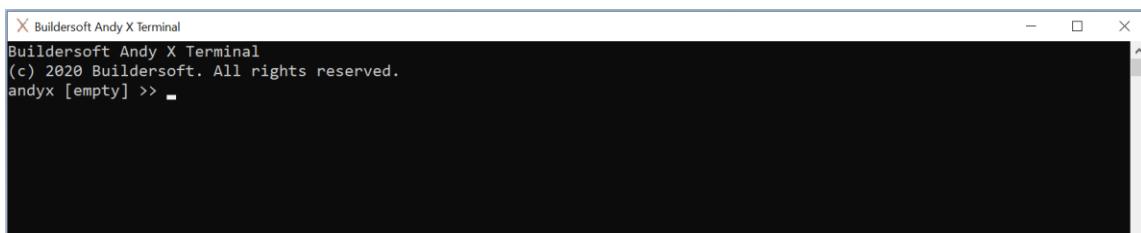
Andy X si distributed streaming platform është i hostuar në Microsoft Azure në mjedisin development i cili gjendet në linkun e dhënë më poshtë:

Host: <https://dev-eu-andyx-buildersoft.azurewebsites.net/> në këtë site mund të shihen të gjithë endpoint-at që i ofron kjo platformë.

Konfigurimi lokal i Andy X mund të bëhet nëpërmjet Docker-it, se si të bëhet konfigurimi lokal i Andy X atë mund ta gjeni në Github [\[10\]](#)

Për konfigurimin e Andy X duhet të ndjeken këta hapa

- Të shkarkohet aplikacioni **Andy X Terminal** nga **Github** dhe të hapet [\[9\]](#)



```
Buildersoft Andy X Terminal
Buildersoft Andy X Terminal
(c) 2020 Buildersoft. All rights reserved.
andyx [empty] >> -
```

Figura 13: Andy X Terminal

- Duke përdorur komandën **node add** të ndjeken hapat për krijimin e një node. Një Node është element që përban informata e nevojshme për lidhje me Host-in e dhënë më lartë.



```
Buildersoft Andy X Terminal
Buildersoft Andy X Terminal
(c) 2020 Buildersoft. All rights reserved.
andyx [empty] >> node add

Write node properties
Name: automatika-dev
Url: https://dev-eu-andyx-buildersoft.azurewebsites.net
Tenant: automatika
Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhdXrvbwF0aWthIiwianRpIjoiYTRkOTFiM2EtMWQxYi00YTvjLTgzZWUtODYyOTAYMjk3MDRhIiwiVGVuYW50SWQiOijkNWFhOTU4My0xMjkkLTQxYmYtYTDjZC05YTMSMmM2NWFjNWQoIcJUZWS5hbniOijJhdXrvbwF0aWthIiwiU2VjdXJpdHlZZXkiOjIyM0NDAwNi1N2IzLTQzODYtYjA4OS00ZDV3NjRkMzFkNDkiLCJleHAiOjE2MTQ1NDI4NTAsImlzcyI6IkJ1aWxkZXJzb2Z0IiwiYXVkJjoiQW5keVgjQ.WExWilKKKu9CEFeh5Fvqym0T4LoTxRxr25rz6ogJoKE-
```

Figura 14: Andy X Terminal – **node add**

- Pasi të bëhet regjistrimi i këtij node, për t'u lidhur me atë node/tenant duhet të ekzekutoni komandën **node select 0**, ose **node list** pastaj y dhe zgjedhet node nga lista.
- Për të krijuar një Token të ri ju nevojiten këto informata emri i tenant-it në rastin tonë **automatika**, tenant id: **d5aa9583-1291-41bf-a7cd-9a392c65ac5d** dhe SecurityKey: **bbe44006-b7b3-4386-b089-4d6764d31d49**, këto informata duhet të

vendosen sikur në figurën në vazhdim dhe mund të kopjohet tokeni nga rezultati i dhënë.

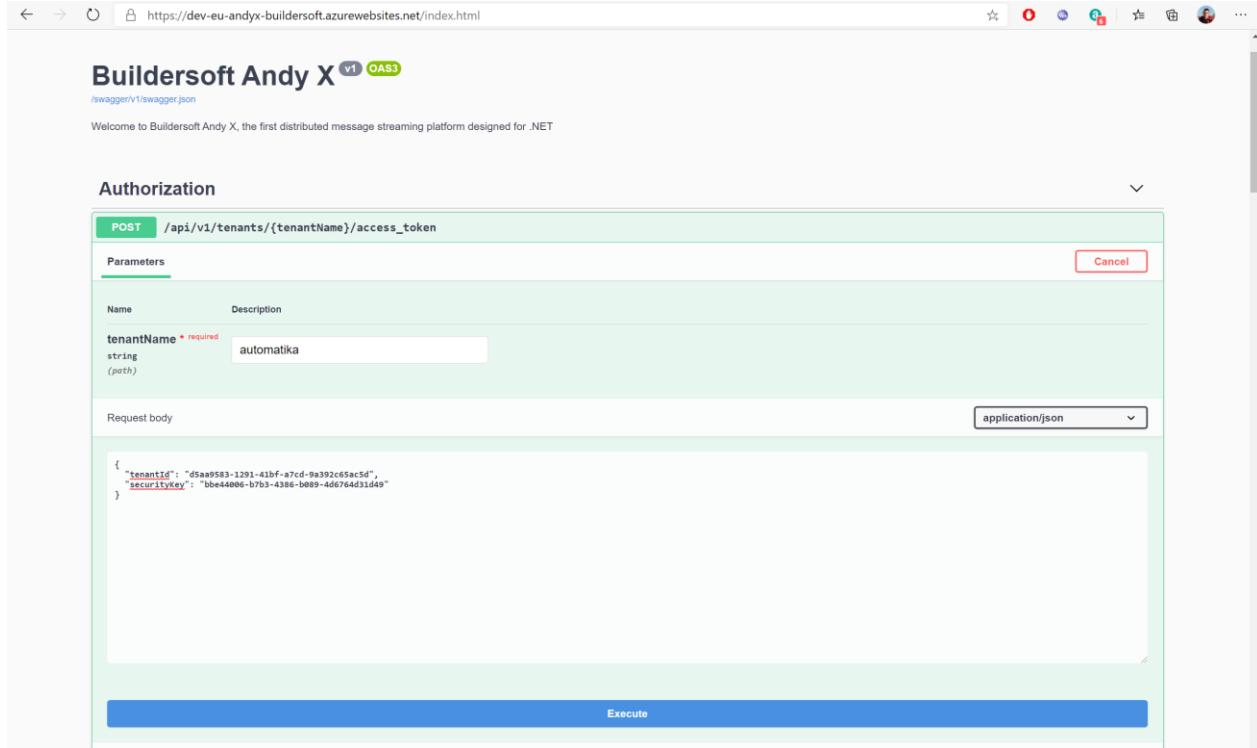


Figura 15: Andy x node mjedisi development – **gjenerimi i tokenit**

```

Code Details
200 Response body
{
    "validTo": "2021-03-06T22:24:00Z",
    "value": "eyJhbGciOiIjUzI1NiIsImScIjpkxV9s...eyJzdWJzIjhdXrbwRoaithiwiianRpIjoiYBzTlHk2UtoTUsoCnNzVklwRhMtkng31ZmMGU50G41iwIVvUrvw5eswQio13knFh0tU4HyExMjKxLTQxVmtyYidjZC05Y1NSMh2NnfJMaQ1LcJUZm5h0nQjOihdkavbwRoaithiwiIuVjpdhIlLzkiO1iYhU0NDawh11N2i1Qz00YtjA4OS06ZDYNjRkmfKnokilC1ehA0jEjMTUUmjk0hDAisImLzcylkJiaNokzJzb28IiwiXvkiIjoiQwskevgfQ...SjFuAdAB57Rm7Hm7vNt2L2L
}
Download

Response headers
content-encoding: gzip
content-type: application/json; charset=utf-8
date: Wed, 03 Mar 2021 01:01 GMT
request-context: appId=cid-v1:65b3470c-3c25-4c00-8961-b5ac1986807b
server: Microsoft-IIS/10.0
transfer-encoding: chunked
vary: Accept-Encoding
x-powered-by: ASP.NET
  
```

Figura 16: Andy x node mjedisi development – **gjenerimi i tokenit rezultati**

- Në Andy X Terminal duke ekzekutuar komandën **tenant view automatika**, mund të lexojmë resurset që ndodhen brenda këtij tenanti, për të krijuar produkt brenda tenantit automatika mundemi duke ekzekutuar komandën **product add automatika/{emri-i-produktit}**.

Pasi që të kemi konfigurar Andy X dhe dy MCU-të të jenë aktiv, mund të fillojmë me dizajnimin dhe zhvillimin e pjesës logjike, në këtë temë është përdor gjuha programuese C

4 Softueri, logjika e mikrokontrollerit dhe kompajlimi

4.1 Ndarja e softuerit

Softueri me të cilin punojnë dy MCU-të e lidhur në internet, është i zhvilluar me gjuhën programuese C# dhe punon brenda sistemit operativ të instaluar në këta MCU i cili është Windows 10 IoT Core. Aplikacioni që quhet **UP.MCU.Edge** është një solution i cili është i ndarë në tri nënpjekte të mëdha të cilat janë **UP.MCU.Edge.App**, **UP.MCU.Edge.Model** dhe **UP.MCU.Edge.Logic**. Ky aplikacion është i publikuar në GitHub [7]

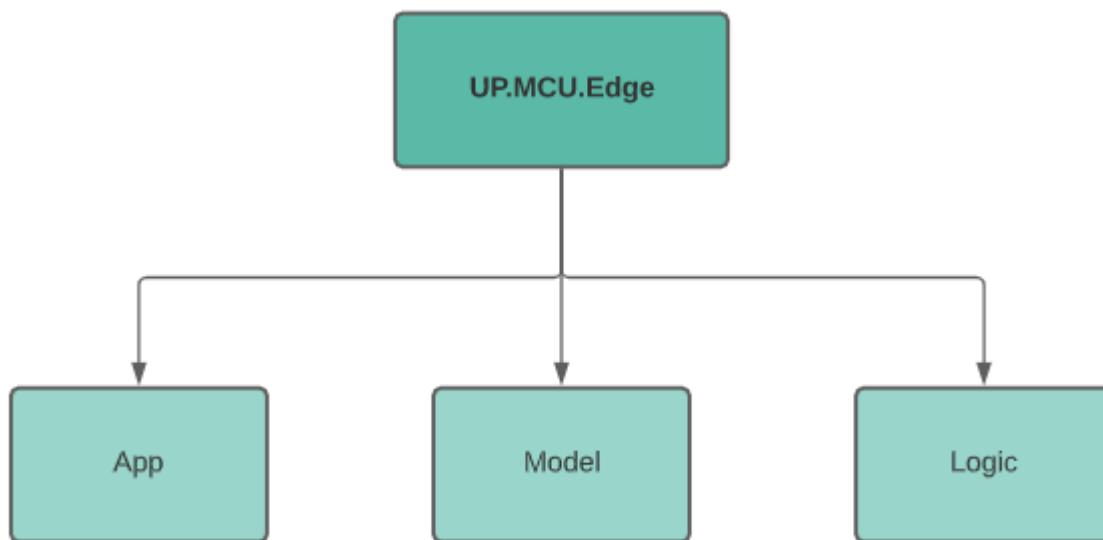


Figura 17: Ndarja e aplikacionit

Pjesa e aplikacionit që eshtë **blloku i parë** në fig.17 **App** në këtë projekt kemi të shkruar logjiken në të cilin bëhet startimi i komponentëve për kryerjen e logjikave brenda MCU-së.

Blloku i dytë në figurën më lartë paraqet modelet mbi të cilat bëhet transmetimi i informatave në mes të dy MCUs.

Blloku i tretë në figurën më lartë paraqet logjiken dhe operacionet që aplikacioni kryen përgjatë komunikimit në mes dy MCUs përmes cloud.

4.1.1 Aplikacioni

Nëse ky aplikacion është duke vepruar në MCU dhe nëse do lidhet nëpërmjet HDMI me një ekran të jashtëm, në të do të paraqitet pamja si në vazhdim. Në figurën 18 shihet dizajnimi i ndërsaçës së aplikacionit i cili është bërë duke përdorur gjuhën deklarative të bazuar në XML që njihet si **XAML**, kjo faqe brenda këtij aplikacioni njihet me emrin **MainPage.xaml**

Softueri, logjika e mikrokontrollerit dhe kompjajlimi

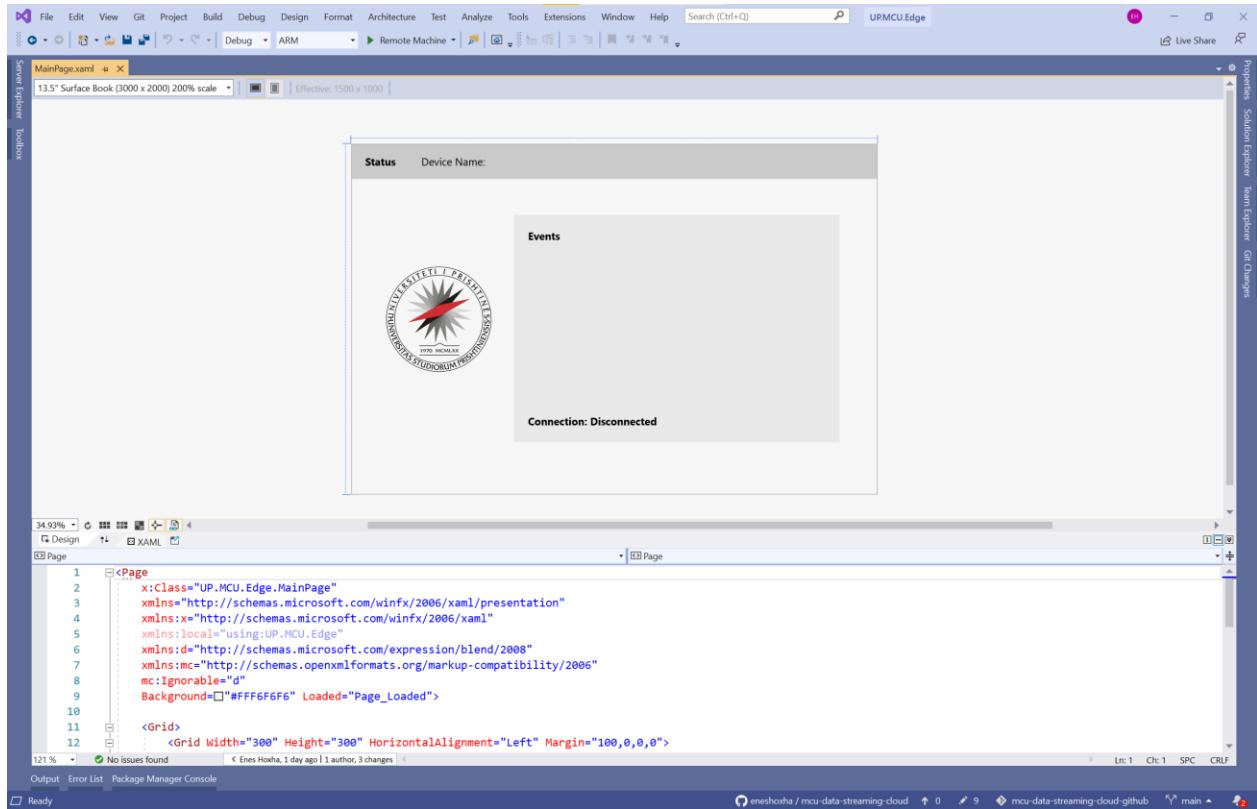


Figura 18: Dizajnimi i ndërfaqës për MCU

Së bashku me pjesën e ndërfaqes ndodhet edhe pjesa e logjikës për këtë dritare e cila është e dhënë në figurën 19.

```
20  namespace UP.MCU.Edge
21  {
22      /// <summary>
23      /// An empty page that can be used on its own or navigated to within a Frame.
24      /// </summary>
25      public sealed partial class MainPage : Page
26      {
27          private string machineName;
28          private AndyXProvider andyXProvider;
29          public MainPage()
30          {
31              this.InitializeComponent();
32          }
33
34          private void Page_Loaded(object sender, RoutedEventArgs e)
35          {
36              machineName = Environment.MachineName;
37              txtDeviceName.Text = $"Device name: {Environment.MachineName}";
38
39              string url = "https://dev-eu-andyx-buildersoft.azurewebsites.net";
40              string tenant = "automatika";
41              string product = "mcu-streaming";
42              string token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwdWIiOiJhdXrvbWF0aWthIiwianRpIjoiYTRkOTFIM2EtMWQxYi00YTVjLTgzzWUtODYyOTAyMjk3MDRhIiwiVGVuYW50SWQiOjIk
43              andyXProvider = new AndyXProvider(url, tenant, product, token);
44              if (andyXProvider.IsAndyXClientConnected() == true)
45              {
46                  txtConnection.Text = $"Connection: Connected";
47                  new BaseService(andyXProvider).UseAndyXServices(machineName);
48              }
49          }
50      }
```

Figura 19: MainPage.cs

Në momentin e startimit të aplikacionit do të thirret eventi **Page_Loaded** (e cila është metodë por në kontekstin e krijimit të saj ka funksionin e një eventi). Marrim emrin e pajisjes të cilin e kemi vendosur kur e kemi instaluar sistemin operativ në MCU. Pastaj kemi inicuar lidhjen me Andy X Node me informatastë cilat i kemi nga konfigurimi i tij. **AndyXProvider** është një klasë e cila bënë lidhjen me AndyX dhe ofron atë gjendjë për shërbimet të cilat do e kryejnë logjikën e rregullimit.

```

1  using Buildersoft.Andy.X.Client;
2  using System;
3  using System.Collections.Generic;
4  using System.Text;
5
6  namespace UP.MCU.Edge.Logic.Providers
7  {
8      public class AndyXProvider
9      {
10         private readonly AndyXClient _andyXClient;
11         private bool isBuild = true;
12         public AndyXProvider(string url, string tenant, string product, string token)
13         {
14             _andyXClient = new AndyXClient(url)
15                 .Tenant(tenant)
16                 .Product(product)
17                 .Token(token);
18
19             _andyXClient.BuildAsync();
20         }
21
22         public AndyXClient GetAndyXClient()
23         {
24             return _andyXClient;
25         }
26
27         public bool IsAndyXClientConnected()
28         {
29             return isBuild;
30         }
31     }
32 }
```

Figura 20: AndyXProvider

Pasi që lidhja të është realizuar do të inicohet **BaseService** i cili pastaj do ti inicoj shërbimet varësisht nga emri i MCU-së që është duke u ekzekutuar kodi. Para se te fillojmë analizen e BaseServices fillimisht duhet krijuar modelet e të dhenave me te cilat do të ndodhë komunikimi.

4.1.2 Modelet e të dhënavë

Modelet e të dhënavë janë pjesa me e rëndësishme e çdo aplikacioni pasi që e tërë logjika e cila shkruhet mbrapa ka të bëjë me përpunimin, transmetimin dhe ruajtjen e të dhënavëve. Pra, në bazë të këtyre modeleve do të bëjmë komunikimin në mes MCU-ve. Në këtë temë kemi dy ndarje të modeleve **Test** dhe **Time** si eksperimente. Këto dy eksperimente në vete kanë nga dy modele *Kërkesë/Përgjigje apo Komand/Ngjarje*.

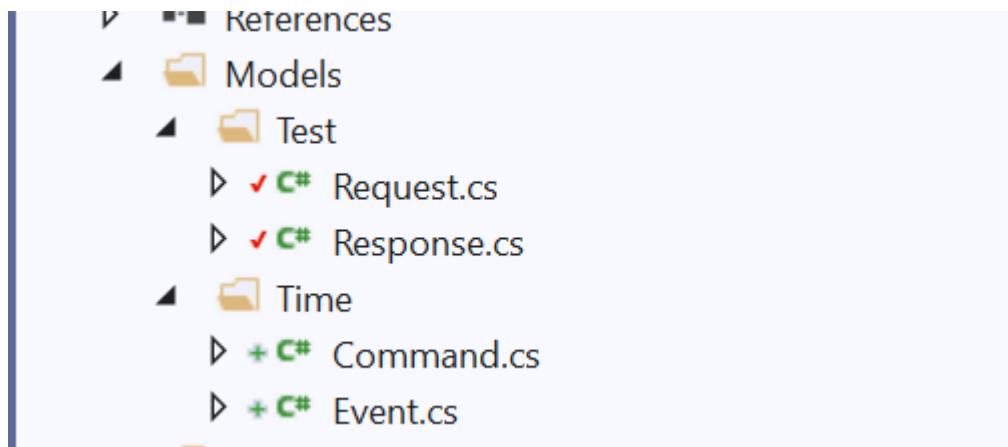


Figura 21: Ndarja e modeleve të eksperimenteve

Të dhënat që kanë brenda modelet e eksperimentit Test janë si në figurën 22, në vazhdim.

```

Request.cs
using System;
using System.Collections.Generic;
using System.Text;

namespace UP.MCU.Edge.Logic.Models.Test
{
    public class Request
    {
        public static string ComponentName = "kontrolleri-test";
        public static string BookName = "komanda-test";

        public double Autonumber { get; set; }
        public string Message { get; set; }
    }
}

Response.cs
using System;
using System.Collections.Generic;
using System.Text;

namespace UP.MCU.Edge.Logic.Models.Test
{
    public class Response
    {
        public static string ComponentName = "kontrolleri-test";
        public static string BookName = "ngjarja-test";

        public double Autonumber { get; set; }
        public string Message { get; set; }
    }
}

```

Figura 22: Modelet Request dhe Response për eksperimentin Test

Sic shihet në figuren 22, pjesa e nënvizuar ComponetName dhe BookName, janë të dhënat që na nevojiten për iniciimin e Andy X Lexuesit dhe Shkruesit. Të dhënat që do të transmetohen nga njeri MCU në tjetrit do të janë **Autonumber** i tipit **double** dhe **Message** i tipit **string**.

Për eksperimenin e dytë të lidhur me llogaritjen e vonesave kohore nga dërgimi i mesazhit e deri tek pranimi i tij janë të emërtuar si **Command** dhe **Event**. Command është modeli me të cilin MCU-01 do të dërgon çdo 1 sekond komand tek MCU-02, të dhënat që këto dy modele kane në vete janë si në figurën 23.

```

Command.cs
1 1. Using System;
2 2. using System.Collections.Generic;
3 3. using System.Linq;
4 4. using System.Text;
5 5. using System.Threading.Tasks;
6
7 6. using UP.MCU.Edge.Logic.Models.Time;
8
9 11 references | 0 changes | 0 authors, 0 changes
10 public class Command
11 {
12     public static string ComponentName = "kontrolleri-kohes";
13     public static string BookName = "komanda-kohe";
14
15     2 references | 0 changes | 0 authors, 0 changes
16     public DateTime Data { get; set; }
17     3 references | 0 changes | 0 authors, 0 changes
18     public TimeSpan Koha { get; set; }
19
20 }
21
Event.cs
1 1. Using System;
2 2. using System.Collections.Generic;
3 3. using System.Linq;
4 4. using System.Text;
5 5. using System.Threading.Tasks;
6
7 6. using UP.MCU.Edge.Logic.Models.Time;
8
9 6 references | 0 changes | 0 authors, 0 changes
10 public class Event
11 {
12     public static string ComponentName = "kontrolleri-kohe";
13     public static string BookName = "ngjarja-kohe";
14
15     1 reference | 0 changes | 0 authors, 0 changes
16     public DateTime Data { get; set; }
17     1 reference | 0 changes | 0 authors, 0 changes
18     public TimeSpan Koha { get; set; }
19
20     1 reference | 0 changes | 0 authors, 0 changes
21     public TimeSpan KohaEPranuar { get; set; }
22     1 reference | 0 changes | 0 authors, 0 changes
23     public double DiferencaNeMilisekonda { get; set; }
24
25 }
26

```

Figura 23: Modelet Command dhe Event për eksperimentin Time

Pas dizajnjimit të modeleve me të cilat do të punojnë, duhet të analizojmë strukturën e shërbimeve brenda logjikës së aplikacionit e cila analizohet në vazhdim.

4.1.3 Logjika

Logjika është pjesë e rëndësishme e çdo aplikacionit pasi që reprezenton komplet mënyrën se si aplikacioni punon. Tek kjo pjese kemi të paraqitur shërbimet (*en. services*) të cilat mundësojnë komunikimin dhe përpunimin e informatave të *dërguara/pranuara*. Struktura e ndarjes së projektit është dhënë si në figurën 24.

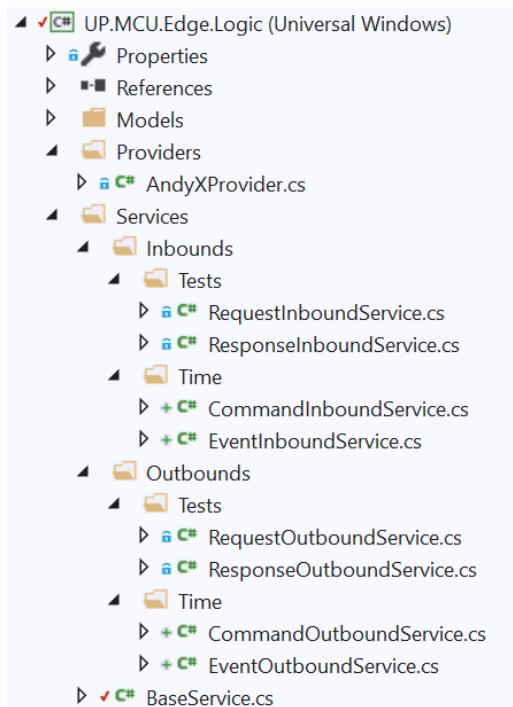


Figura 24: Struktura e ndarjes së klasave brenda logjikës së aplikacionit

Nga figura më lartë shihet ndarja e eksperimenteve nën folderin **Services**. Kemi dy lloje të shërbimeve të cilat janë **Inbound** dhe **Outbound**. **Inbound Sherbimet** kanë për detyrë që të pranojnë informacionin nga MCU-të apo shërbimet tjera. **Outbound Sherbimet** kanë për detyrë që të dërgojnë informacion të përpunuar tek MCU-të apo shërbimet tjera për përpunim të mëtutjeshëm.

E thënë ndryshe, Inbound shërbimet bëjnë leximin e informatave nga Andy X, ndërsa Outbound shërbimet bëjnë shkrimin e informatave në Andy X, gjithëashtu në figuren 24 mund të shihet se ndodhet edhe një shërbim tjetër që njihet si **BaseService**, ky shërbim inicohet nga vet aplikacioni i dhënë në figuren 19 dhe ka për detyrë të bëjë inicializimin e Inbound dhe Outbound shërbimeve.

RequestInboundService në vete mbanë logjikën e cila lexon në librin e kërkuesave i cili është i vendosur në Modelin **Request** me emër të variables **BookName**. I cili pranon një kërkuesë për një njësi kohore dhe lexon informatën në momentin e arritjes në libër në Andy X. Çdo kërkuesë e cila arrin në këtë reader ngritë një event që njihet si **Integration**. Eventi Integration ndodhet brenda klasës *Source* i cila është pjesë e *Andy X Toolkit*. Ky shërbim në momentin e pranimit të kërkuesës do të pauzojë për 10 sekonda dhe do të dërgojë përgjigjën me të njejtat informata të pranuara tek MCU-ja tjetër.

```

9  namespace UP.MCU.Edge.Logic.Services.Inbounds.Tests
10 {
11     2 references | Enes Hoxha, 1 day ago | 1 author, 3 changes
12     public class RequestInboundService
13     {
14         private readonly Source<Models.Test.Request> source;
15         private readonly ResponseOutboundService _responseOutboundService;
16         public RequestInboundService(AndyXClient andyXClient)
17         {
18             var reader = new Reader<Models.Test.Request>(andyXClient.GetClient()
19                 .Component<Models.Test.Request>.ComponentName)
20                 .Book(<Models.Test.Request>.BookName)
21                 .ReaderType<Buildersoft.Andy.X.Client.Model.ReaderTypes>(Buildersoft.Andy.X.Client.Model.ReaderTypes.Exclusive)
22                 .ReaderName("request-inbound-up-edge-mcu")
23                 .ReaderAs<Buildersoft.Andy.X.Client.Model.ReaderAs>(Buildersoft.Andy.X.Client.Model.ReaderAs.Subscription)
24                 .Build();
25
26             _responseOutboundService = new ResponseOutboundService(andyXClient);
27
28             source = new Source<Models.Test.Request>(reader)
29                 .Configure(new Buildersoft.Andy.X.Streams.Settings.SourceConfigurationSettings()
30                     .Throttle(1, 1));
31
32             source.Integration += Source_Integration;
33             source.InitializeReader();
34         }
35         1 reference | Enes Hoxha, 1 day ago | 1 author, 2 changes
36         private async void Source_Integration(object sender, Buildersoft.Andy.X.Streams.Model.Data<Models.Test.Request> e)
37         {
38             Thread.Sleep(new TimeSpan(0, 0, 10));
39             await _responseOutboundService.WriteResponse(new Models.Test.Response() { Message = e.RawData.Message, Autonumber = e.RawData.Autonumber });
40         }
41     }

```

Figura 25: Struktura e kodit të logjikës së RequestInboundService

ResponseInboundService punon në MCU-02, në vete mbanë logjikën e cila lexon nga libri i përgjigjeve të cilat janë të vendosura me modelin **Response** nga ResponseOutboundService. Këto përgjigje pranohen si **Kërkesa** nga *RequestInboundService* prej eventit Integration. Ky shërbim në momentin e pranimit të përgjiges do të presë për 10 sekonda dhe do të shkruajë një kërkuesë të

re tek RequestInboundService i cili ndodhet në MCU-01, me mesazhin e njejtë por duke e rritur variablën Autonumber për 1.

```

9  namespace UP.MCU.Edge.Logic.Services.Inbounds.Tests
10 {
11     public class ResponseInboundService
12     {
13         private readonly Source<Response> source;
14         private readonly RequestOutboundService _requestOutboundService;
15
16         public ResponseInboundService(AndyXClient andyXClient)
17         {
18             var reader = new Reader<Response>(andyXClient.GetClient())
19                 .Component(Response.ComponentName)
20                 .Book(Response.BookName)
21                 .ReaderType(Buildersoft.Andy.X.Client.Model.ReaderTypes.Exclusive)
22                 .ReaderName("response-inbound-up-edge-mcu")
23                 .ReaderAs(Buildersoft.Andy.X.Client.Model.ReaderAs.Subscription)
24                 .Build();
25
26             _requestOutboundService = new RequestOutboundService(andyXClient);
27
28             _requestOutboundService.WriteRequest(new Request() { Message = "IncrementNumber", Autonumber = 1 });
29
30             source = new Source<Response>(reader)
31                 .Configure(new Buildersoft.Andy.X.Streams.Settings.SourceConfigurationSettings())
32                 .Throttle(1, 1);
33
34             source.Integration += Source_Integration;
35             source.InitializeReader();
36         }
37
38         private async void Source_Integration(object sender, Data<Response> e)
39         {
40             Thread.Sleep(new TimeSpan(0, 0, 10));
41             await _requestOutboundService.WriteRequest(new Request() { Message = e.RawData.Message, Autonumber = ++e.RawData.Autonumber });
42         }
43     }
44 }
```

Figura 26: Struktura e kodit të logjikës së ResponseInboundService

Pjesa e nënvizuar me te gjelbërt në momentin e inicimit dërgon kërkesën e parë tek MCU-01 dhe kështu fillon komunikimi i pa ndërprerë çdo 10 sekonda në mes dy MCU-ve.

RequestOutboundService iniciohet nga ResponseInboundService për eksperimentin **Test** i cili bënë shkrimin/dërgimin e informacionit deri në destinacion që është Andy X Libri. Ky shërbim përdorë objektin e tipit Writer për të shkruar informacionin e tipit Request. Struktura e kodit të këtij shërbimi është dhënë në figurën në vazhdim.

```

7  namespace UP.MCU.Edge.Logic.Services.Outbounds.Tests
8  {
8.5    2 references | Enes Hoxha, 9 days ago | 1 author, 1 change
9      public class RequestOutboundService
10     {
11         private Writer<Models.Test.Request> writer;
12         public RequestOutboundService(AndyXClient andyXClient)
13         {
14             writer = new Writer<Models.Test.Request>(andyXClient.GetClient())
15                 .Component(Models.Test.Request.ComponentName)
16                 .Book(Models.Test.Request.BookName)
17                 .Schema(Buildersoft.Andy.X.Client.Model.SchemaTypes.Json)
18                 .WriterType(Buildersoft.Andy.X.Client.Model.WriterTypes.StreamAndStore)
19                 .Build();
20         }
21
22         2 references | Enes Hoxha, 9 days ago | 1 author, 1 change
23         public async Task<Guid> WriteRequest(Models.Test.Request request)
24         {
25             return await writer.WriteAsync(request);
26         }
27     }
28 }
```

Figura 27: Struktura e kodit të logjikës së RequestOutboundService

ResponseOutboundService inicohet nga RequestInboundService për eksperimentin **Test** i cili bënë shkrimin/dërgimin e përgjigjës në destinacionin përkatës. Ky shërbim njëlojë sikur RequestOutboundService përdorë objektin e tipit Writer për të shkruar informacionin e tipit Response. Struktura e kodit të këtij shërbimi është dhënë në figurën në vazhdim.

```
7   namespace UP.MCU.Edge.Logic.Services.Outbounds.Tests
8   {
9     public class RequestOutboundService
10    {
11      private Writer<Models.Test.Request> writer;
12      public RequestOutboundService(AndyXClient andyXClient)
13      {
14        writer = new Writer<Models.Test.Request>(andyXClient.GetClient())
15          .Component(Models.Test.Request.ComponentName)
16          .Book(Models.Test.Request.BookName)
17          .Schema(Buildersoft.Andy.X.Client.Model.SchemaTypes.Json)
18          .WriterType(Buildersoft.Andy.X.Client.Model.WriterTypes.StreamAndStore)
19          .Build();
20      }
21
22      public async Task<Guid> WriteRequest(Models.Test.Request request)
23      {
24        return await writer.WriteAsync(request);
25      }
26    }
27 }
```

Figura 28: Struktura e kodit të logjikës së ResponseOutboundService

CommandInboundService është shërbimi i cili pranon komandat për procesim dhe është pjesë e eksperimentit të dytë **Time** dhe ekzekutohet nga **MCU-02**. Struktura e kodit është e përafërt me atë *Requset/ResponseInboundService*. Ajo që e dallon është modeli i të dhënavë që e pranon i cili është i tipit *Command*. Logjika e rregullimit në momentin e pranimit të kërkesës nga BaseService, i cili i inicion kërkesat nga *MCU-01* është e shkruar në ngjarjen (*en. event*) Integration i cili thotë kështu: Në momentin e pranimit të kërkesës nga MCU-01 nëpërmjet Cloud, lexohet koha e pranimit të informatës dhe konstrukton objektin e tipit Event të cilin i shoqërohen të dhënat e pranuara, vendos kohën e pranimit dhe kalkulon diferençën kohore të cilën do e analizojmë si **vonesa kohore**. Pasi, që këto veprime të janë përfunduar nëpërmjet objektit të tipit **EventOutboundService** bëhet transmetimi i kësaj ngjarje me një libër në Andy X. Struktura e kodit për CommandInboundService është paraqitur në figurën në vazhdim.

```

11  namespace UP.MCU.Edge.Logic.Services.Inbounds.Time
12  {
13      2 references | 0 changes | 0 authors, 0 changes
14      public class CommandInboundService
15      {
16          private readonly Source<Command> source;
17          private readonly EventOutboundService _eventOutboundService;
18          1 reference | 0 changes | 0 authors, 0 changes
19          public CommandInboundService(AndyXClient andyXClient)
20          {
21              var reader = new Reader<Command>(andyXClient.GetClient())
22                  .Component(Command.ComponentName)
23                  .Book(Command.BookName)
24                  .ReaderType(Buildersoft.Andy.X.Client.Model.ReaderTypes.Exclusive)
25                  .ReaderName("command-time-inbound-up-edge-mcu")
26                  .ReaderAs(Buildersoft.Andy.X.Client.Model.ReaderAs.Subscription)
27                  .Build();
28
29              _eventOutboundService = new EventOutboundService(andyXClient);
30
31              source = new Source<Command>(reader)
32                  .Configure(new Buildersoft.Andy.X.Streams.Settings.SourceConfigurationSettings())
33                  .Throttle(1, 1);
34
35              source.Integration += Source_Integration;
36              source.InitializeReader();
37
38          }
39
40          1 reference | 0 changes | 0 authors, 0 changes
41          private async void Source_Integration(object sender, Buildersoft.Andy.X.Streams.Model.Data<Command> e)
42          {
43              TimeSpan kohaEPranuar = DateTime.Now.TimeOfDay;
44              Event @event = new Event()
45              {
46                  Data = e.RawData.Data,
47                  Koha = e.RawData.Koha,
48                  KohaEPranuar = kohaEPranuar,
49                  DiferencaNeMilisekonda = (kohaEPranuar - e.RawData.Koha).TotalMilliseconds
50              };
51              await _eventOutboundService.WriteEvent(@event);
52          }
53      }
54  }

```

Figura 29: Struktura e kodit të logjikës së CommandInboundService

CommandOutboundService është shërbimi i cili ka për detyrë që të transmetojë komandat nga MCU-01 në MCU-02. Këto komanda do të dërgohen çdo 1 sekond nga **BaseService**. Struktura e kodit për këtë shërbim është dhënë në figurën 30.

EventOutboundService është shërbimi i cili ka për detyrë që të transmetojë ngjarjen e zhvilluar nga komanda e pranuar nga MCU-01 në MCU-02. Ky shërbim transmeton ngjarjen nga *CommandInboundService*. EventOutboundService ekzekutohet në MCU-02. Struktura e kodit për këtë shërbim është dhënë në figurën 31.

```
9   namespace UP.MCU.Edge.Logic.Services.Outbounds.Time
10  {
11    public class CommandOutboundService
12    {
13      private readonly Writer<Command> writer;
14      public CommandOutboundService(AndyXClient andyXClient)
15      {
16        writer = new Writer<Command>(andyXClient.GetClient())
17          .Component(Command.ComponentName)
18          .Book(Command.BookName)
19          .Schema(Buildersoft.Andy.X.Client.Model.SchemaTypes.Json)
20          .WriterType(Buildersoft.Andy.X.Client.Model.WriterTypes.StreamAndStore)
21          .Build();
22      }
23
24      public async Task<Guid> WriteCommand(Command command)
25      {
26        return await writer.WriteAsync(command);
27      }
28    }
29 }
```

Figura 30: Struktura kodit te logjikes se CommandOutboundService

```
6   namespace UP.MCU.Edge.Logic.Services.Outbounds.Time
7   {
8    public class EventOutboundService
9    {
10      private readonly Writer<Event> writer;
11      public EventOutboundService(AndyXClient andyXClient)
12      {
13        writer = new Writer<Event>(andyXClient.GetClient())
14          .Component(Event.ComponentName)
15          .Book(Event.BookName)
16          .Schema(Buildersoft.Andy.X.Client.Model.SchemaTypes.Json)
17          .WriterType(Buildersoft.Andy.X.Client.Model.WriterTypes.StreamAndStore)
18          .Build();
19      }
20
21      public async Task<Guid> WriteEvent(Event @event)
22      {
23        return await writer.WriteAsync(@event);
24      }
25    }
26  }
```

Figura 31: Struktura e kodit të logjikës së EventOutboundService

BaseService është shërbimi bazë i cili inicohet nga aplikacioni në momentin e startimit të MCU-së. Ky shërbim mbanë logjikën e dy MCU-ve dhe i dallon ata në bazë të emrit të tyre. Ky shërbim gjithëashtu ka një Kohëmatës (*en. Timer*) i cili çdo 1 sekondë dërgon datën dhe orën e saktë nga **MCU-01** në **MCU-02** nëpërmjetë cloud. Logjika e tij është e ndarë në dy pjesë, nëse MCU ka emrin **bs-andy-mcu-01**, ky shërbim do të inicoj **RequestInboundService** dhe kohëmatësin për dërgim të komandave nëpërmjetë **CommandOutboundService**. Në rastin kur ky service do të inicohet në MCU-në me emër **bs-andy-mcu-02** atëherë do të inicoj këto shërbime **ResponseInboundService** dhe **CommandInboundService**. MCU-02 përmbanë dy shërbime të cilat pranojnë dhe procesojnë informacion, ajo që mund të perfundojmë këtu është se këto dy shërbime janë të izoluara nga njëra-tjetra dhe logjika e tyre është kompletë ndryshe. Struktura e kodit të këtij shërbimi është dhënë në figurën në vazhdim.

```

namespace UP.MCU.Edge.Logic.Services
{
    1 reference | Enes Hoxha, 1 day ago | 1 author, 1 change
    public class BaseService
    {
        private Timer sendCommandsToMcuTimer;
        private readonly AndyXProvider _andyXProvider;
        private CommandOutboundService commandOutboundService;

        1 reference | Enes Hoxha, 1 day ago | 1 author, 1 change
        public BaseService(AndyXProvider andyXProvider)
        {
            _andyXProvider = andyXProvider;

            sendCommandsToMcuTimer = new Timer();
            sendCommandsToMcuTimer.Interval = new TimeSpan(0, 0, 1).TotalMilliseconds;
            sendCommandsToMcuTimer.Elapsed += SendCommandsToMcuTimer_Elapsed;
            sendCommandsToMcuTimer.Stop();
        }

        1 reference | Enes Hoxha, 1 day ago | 1 author, 1 change
        public void UseAndyXServices(string machineName)
        {
            if (machineName == "bs-andy-mcu-01")
            {
                RequestInboundService requestInboundService = new RequestInboundService(_andyXProvider.GetAndyXClient());
                commandOutboundService = new CommandOutboundService(_andyXProvider.GetAndyXClient());
                sendCommandsToMcuTimer.Start();
            }
            else
            {
                ResponseInboundService responseInboundService = new ResponseInboundService(_andyXProvider.GetAndyXClient());
                CommandInboundService commandInboundService = new CommandInboundService(_andyXProvider.GetAndyXClient());
            }
        }

        1 reference | 0 changes | 0 authors, 0 changes
        private async void SendCommandsToMcuTimer_Elapsed(object sender, ElapsedEventArgs e)
        {
            await commandOutboundService.WriteCommand(new Models.Time.Command()
            {
                Data = DateTime.Now,
                Koha = DateTime.Now.TimeOfDay
            });
        }
    }
}

```

Figura 32: Struktura e kodit të logjikës së BaseService

4.2 Lidhja e MCU-ve dhe testimi i aplikacionit

Për të ju qasur MCU-ve fillimisht duhet ti lidhim lokalisht me kabllo të internetit UTP, me një modem/router në të cilin është i lidhur edhe kompjuteri.

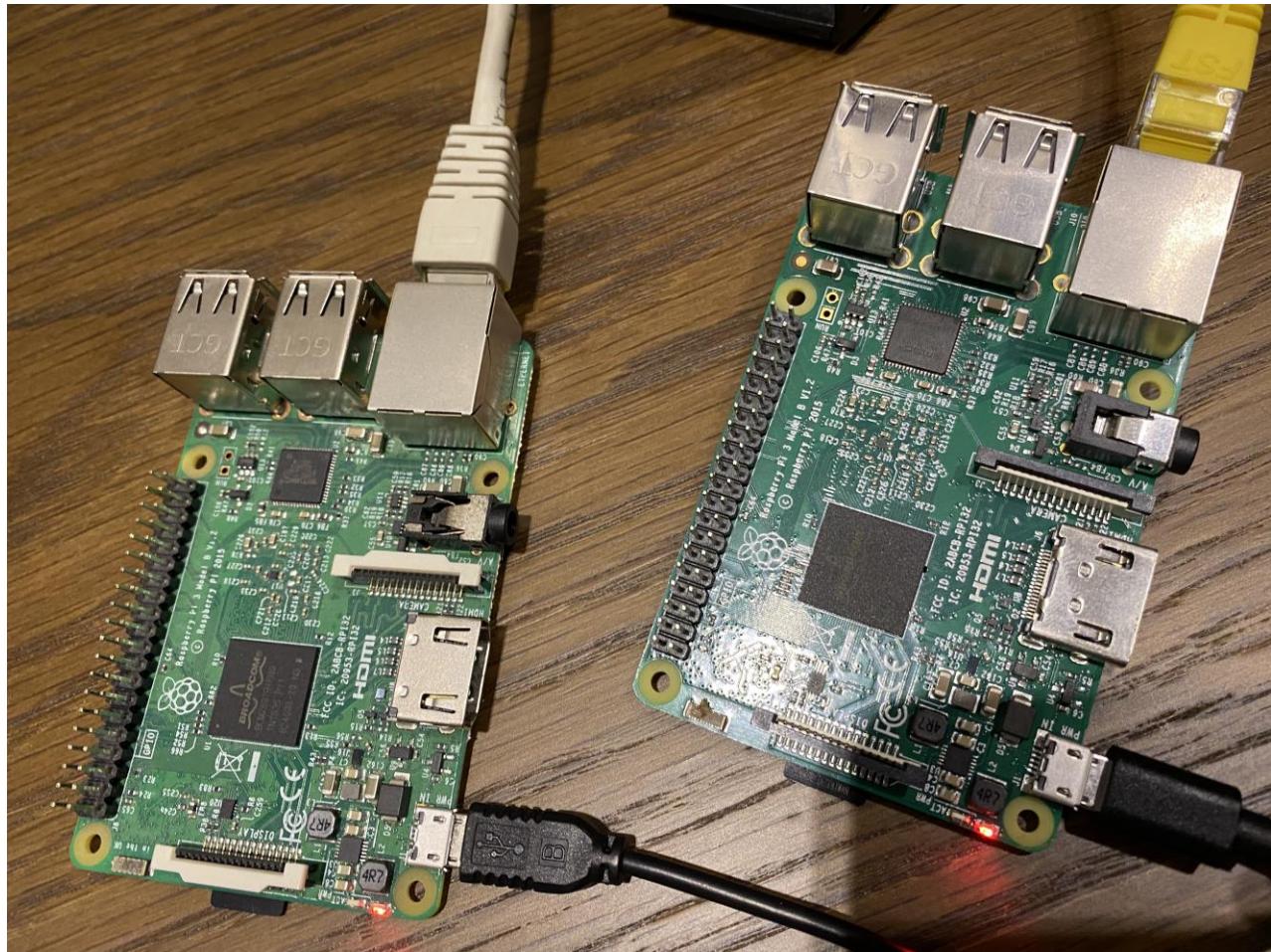


Figura 33: Lidhja e MCU-ve

Për ti marr Ip-Adresat këtyre dy MCU-ve, hapim aplikacionin Windows IoT Core Dashboard dhe tek My devices do të paraqitet në liste dy MCU-te e lidhur në rrjetë si në figurën në vazhdim ku shihen edhe Ip-Adresat të cilat i kanë marr së bashku me emrat e tyre.

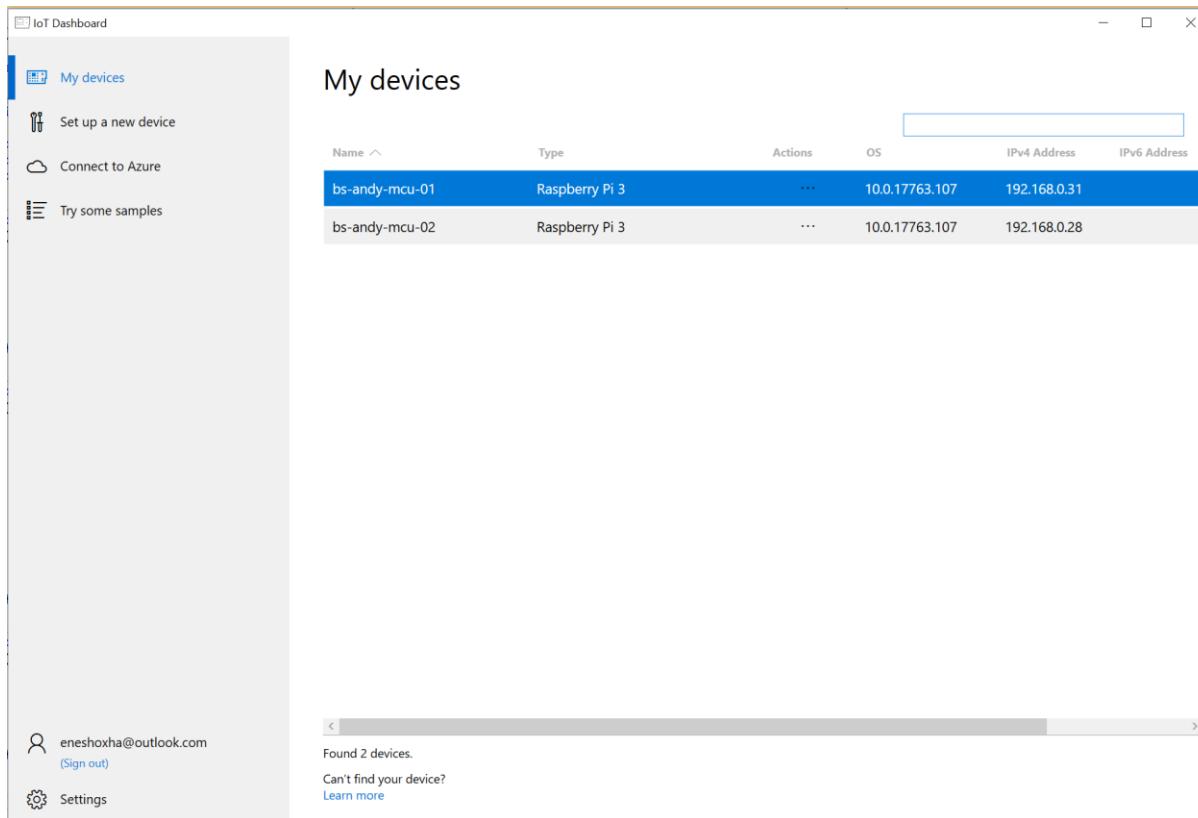


Figura 34: Lista e MCU-ve aktiv

4.2.1 Testimi i aplikacionit

Për testimin e aplikacionit në dy MCU-të në këtë temë kemi përdorur Visual Studio-në. Pasi që të hapim aplikacionin nëpërmjet Visual Studio-së duhet të bëjmë konfigurimin e MCU-ve si në vazhdim. Pasi që MCU-të janë të lidhur në rrjet lokale me kompjuterin në të cilin është hapur projekti me Visual Studio. Do të përdorim opsjonin për instalim të aplikacionit në Pajisje në Distancë (*en. Remote Machine*). Duhet siguruar që kemi zgjedhë arkitekturën ARM (pasi që Raspberry Pi është një *ARM mikrokontroller*) dhe zgjedhim opsjonin Remote Machine, më pastaj duhet të hapim Parametrat e Aplikacionit si në figurën 34.

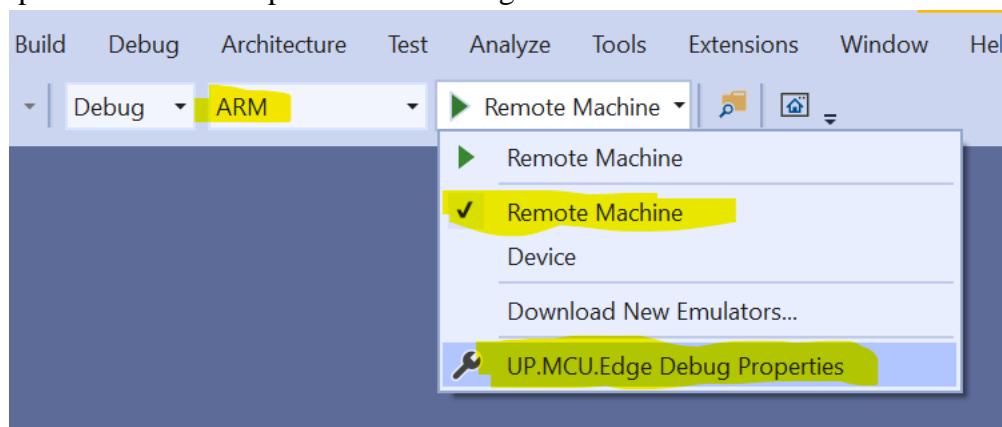


Figura 34: Konfigurimi i Remote Machine

Pasi që të hapet dritarja Properties duhet të klikohet në butonin Gjej (*en. Find*) dhe të klikohet në MCU-në përkatës, në rastin tonë **bs-andy-mcu-01**, pastaj klikohet në butonin Zgjedh (*en. Select*). Për të vazhduar testimin e aplikacionit klikojmë në Remote Machine si në figurën 34.

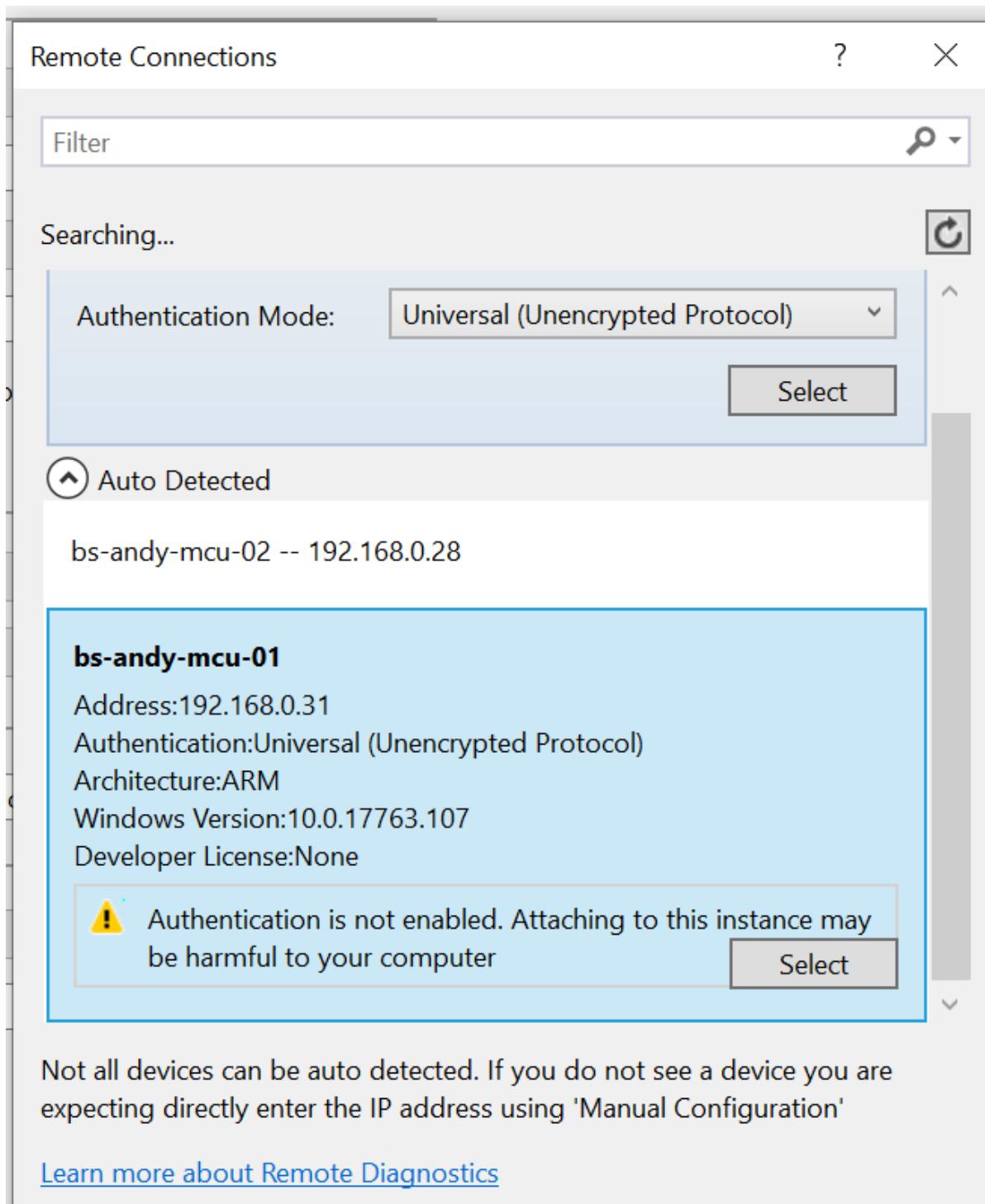


Figura 35: Konfigurimi i Remote Machine – Zgjedhja e MCU-së

4.3 Aplikacioni kompjuterik për lexim të të dhënave gjatë transmetimit

Pasi që të dhënat që dërgohen dhe pranohen nga mikrokontrollerët duke perdorur sistemet e shpërndara të transmetimit, qasja në ato informata pa e prishur strukturën komunikuese apo shfaqjes së ndonjë vonese gjatë komunikimit në mes mikrokontrollerëve është e mundshme edhe nga programe tjera kompjuterike që kanë qasje në internet. Thjeshtë mjafton që të dihen informatat për lidhje me sistemin e shpërndarë për transmetim dhe mund ti qasen këtyre informacioneve. Për të bërë paraqitjen grafike dhe vonesave kohore që shfaqen gjatë rregullimit digjital me komunikim nëpërmjet cloud është krijuar një program kompjuterik duke përdorur gjuhën programuese C# me shablonin (*en. template*) **WPF** (*Windows Presentation Foundation*). Gjithështu edhe ky program së bashku me kodin e tij është publikuar në GitHub. Struktura dhe ndarja e kodit në këtë aplikacionë është mjaftë e thjeshtë, në të gjendet shërbimi **EventInboundService** i cili i lexon ngjarjet të cilat transmetohen nga **MCU-02**. Struktura e kodit është mjaftë e ngjashme me ate të CommandInboundService e cila është treguar në figurën 29, si logjikë në këtë shërbim kemi vizatimi i diagramit të vonesave kohore e cila llogaritet nga MCU-02.

```

namespace UP.MCU.RealtimeReporting.Services.Time
{
    2 references
    public class EventInboundService
    {
        private readonly Source<Event> source;
        private readonly Chart vonesaKoheore;
        private List<KeyValuePair<string, int>> chartDataList;

        1 reference
        public EventInboundService(AndyXClient andyXClient, Chart vonesaKoheore)
        {
            this.vonesaKoheore = vonesaKoheore;

            chartDataList = new List<KeyValuePair<string, int>>();
            ((LineSeries)vonesaKoheore.Series[0]).ItemsSource = chartDataList;

            var reader = new Reader<Event>(andyXClient.GetClient())
                .Component(Event.ComponentName)
                .Book(Event.BookName)
                .ReaderType(Buildersoft.Andy.X.Client.Model.ReaderTypes.Exclusive)
                .ReaderName("reporting-reader-vonesa-kohore")
                .ReaderAs(Buildersoft.Andy.X.Client.Model.ReaderAs.Subscription)
                .Build();

            source = new Source<Event>(reader)
                .Configure(new Buildersoft.Andy.X.Streams.Settings.SourceConfigurationSettings())
                .Throttle(1, 1);

            source.Integration += Source_Integration;
            source.InitializeReader();
        }
        int k = 0;
        1 reference
        private void Source_Integration(object sender, Data<Event> e)
        {
            k++;
            chartDataList.Add(new KeyValuePair<string, int>(k.ToString(), Convert.ToInt32(e.RawData.DiferancaNeMilisekonda)));
            if (chartDataList.Count > 30)
                chartDataList.RemoveAt(0);

            ((LineSeries)vonesaKoheore.Series[0]).ItemsSource = null;
            ((LineSeries)vonesaKoheore.Series[0]).ItemsSource = chartDataList;
        }
    }
}

```

Figura 36: Struktura e kodit të logjikës të EventInboundService

Ndërfaqja e këtij programi është paraqitur në figurën 37 në vazhdim, e cila është mjaftë e thjeshtë dhe në qendër të kësaj dritare paraqitet digrami me vonesën kohore nga MCU-02 çdo 1 sekond së bashku me totalen dhe listën e vlerave të lexuara nga MCU-02.

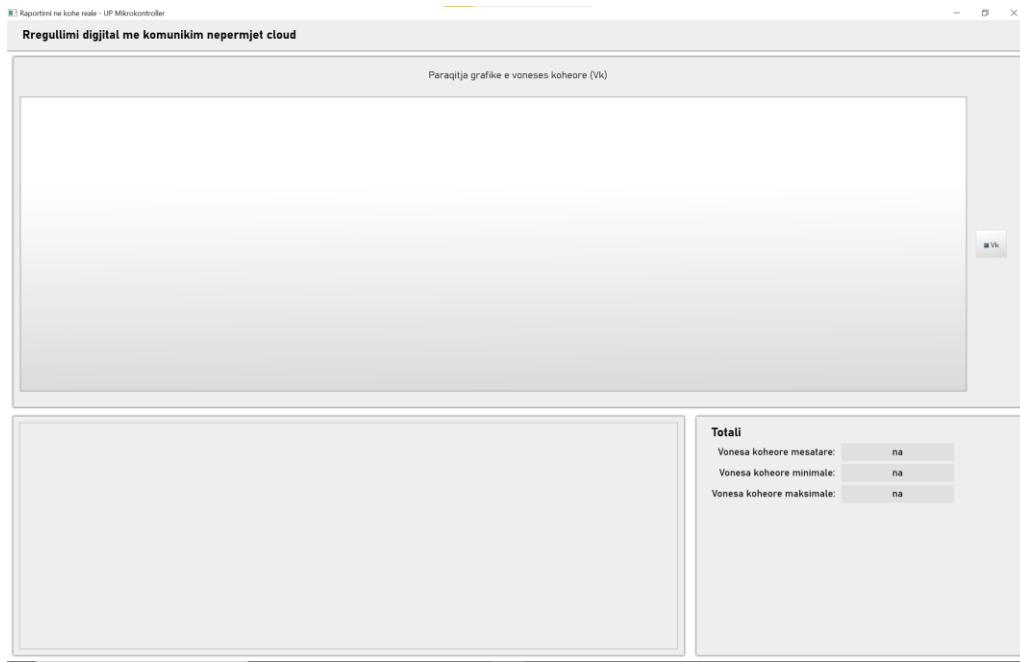


Figura 37: Ndërfaqja e programit kompjuterik për raportim

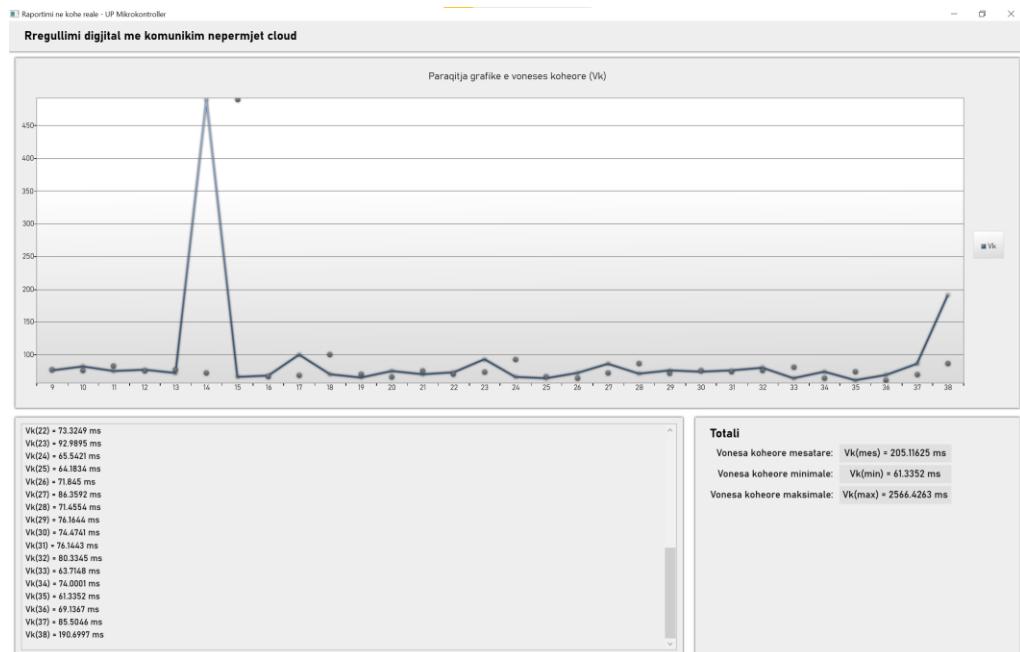


Figura 38: Ndërfaqja e programit kompjuterik për raportim duke lexuar

5 Analiza e eksperimenteve dhe rezultatet dalëse

Gjatë punës profesionale por edhe në jetën e përditëshme gjithmonë do të ketë rezultate dalëse, të cilat varen drejtpërdrejtë nga puna që është bërë në drejtimin e caktuar. Edhe qëllimi i realizimit të këtyre eksperimenteve që do analizohen në vazhdim do të na tregojnë disa rezultate mbi të cilat mund të nxjerrim disa përfundime.

5.1 Matja e vonesës kohore

Pjesa më e rëndësishme gjatë rregullimit digjital me komunikim sidomos nëpërmjet cloud është **vonesa kohore**. Vonesat kohore paraqiten si shkak që pajisjet nuk janë të lidhura drejtpërdrejtë me njëra-tjetrën por ato arrijnë të komunikojnë nëpërmjet internetit. Sot ekzistojnë shumë mënyra të ndryshme për komunikim nëpërmjet internetit, atë që në kemi përzgjedhë për këtë temë janë sistemet e shpërndara për transmetim. Këto vonesa nuk do të mund të eleminohen por mund të janë të menaxhueshme. Dalja e rrjetit 5G do të lejoje që numri i pajisjeve të lidhura në rrjetë të rritet dhe të ofrojë stabilitet të lidhjes gjatë transmetimit të të dhënave përgjatë pajisjeve të ndryshme.

Matja e vonesës kohore gjatë komunikimit të dy MCU-ve të lidhur me njëri-tjetrin nëpërmjet internetit është realizuar në këtë mënyrë. Mikrokontrolleri 01 çdo 1 sekond do të dërgojë mesazhe në mikrokontrollerin 02. Mesazhin të cilin e pranon mikrokontrolleri 02 është:

```
{  
    data: [data e momentit të dërgimit te mesazhit];  
    ora: [momenti kohorë i dërgimit te mesazhit];  
}
```

Pasi që mikrokontrolleri 02 të pranojë mesazhin ai do të ekzekutojë një logjikë e cila krahason momentin kohor **t** të **dërgimit** me atë të **pranimit**. Kjo diferencë e cila në rastin normal do të jetë milisekondëshe, prapë nëpërmjet cloud mikrokontrolleri 02 do e transmetojë tek programi kompjuterik i cili bënë shfaqjen grafike e kësaj vonesë.

Ajo është përmendet që këto vonesa që shfaqen gjatë transmetimeve nëpërmjet internetit varen shumë edhe nga stabiliteti dhe shpejtësia e internetit që e ofron ofruesi (*en. provider*) i internetit. Matjet e vonesës kohore janë bërë në dy ofruesë të ndryshëm të internetit njëri **stabil** ndërsa tjetri **jo-stabil**. Gjithesejë janë shkëmbyer 500 meshazhe në mes të dy mikrokontrollerve, ndërsa llogaritja e mesatares së vonesës kohore është bërë çdo 100 mesazhe të shkëmbyera. Në vazhdim është dhënë diagrami për rregullimin me komunikim për llogaritjen e vonesës kohore.

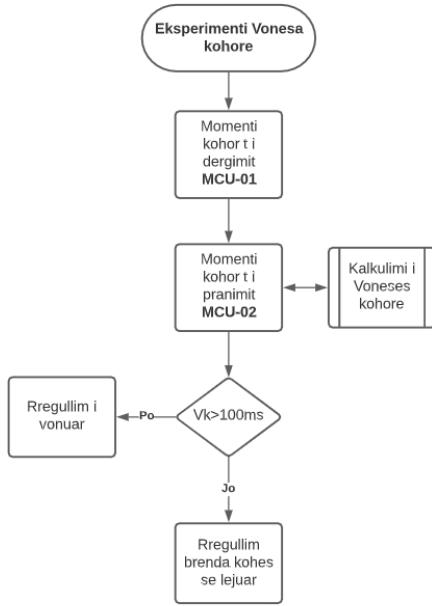


Figura 39: Diagrami për rregullim eksperimenti i parë

Sic shihen edhe në diagramin e dhënë më lartë në figurën 39, për rregullim të një sistemi në mes dy MCU-ve për këtë eksperiment kemi rregullim nëse vonesa kohore është më e vogël se 100 ms.

5.2 Rezultatet nga matja e vonesës kohore në internet stabil

5.2.1 Vonesa kohore në 100 matje

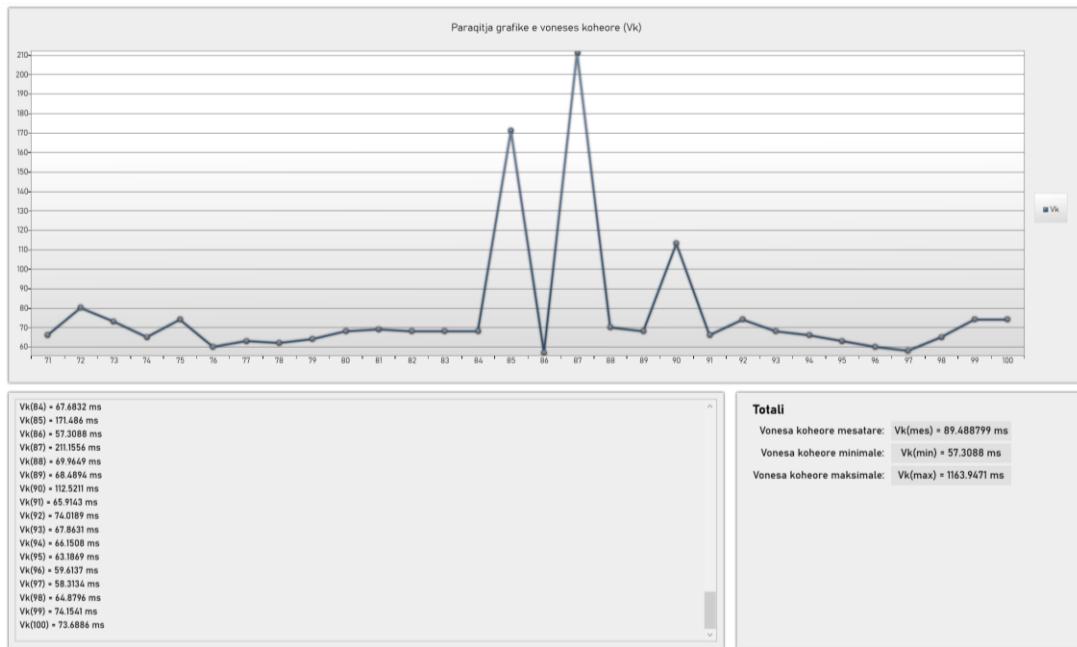


Figura 40: Grafiku i vonesës kohore për 100 matje

Vonesa kohore mesatare Vk(mes) = 89.488799 ms	Vonesa kohore minimale Vk(min) = 57.3088 ms	Vonesa kohore maksimale Vk(max) = 1163.9471 ms
---	---	--

Nga grafiku i dhënë në figurën 39 shihet se nga 100 mesazhet e transmetuara vonesa kohore është e ndryshueshme në mes asaj minimale që është **57ms** deri tek ajo maksimale **1163 ms** ndërsa vonesa kohore mesatare është nën vlerën **100 ms** pra **89 ms**.

5.2.2 Vonesa kohore në 200 matje

Vonesa kohore mesatare Vk(mes) = 85.60283 ms	Vonesa kohore minimale Vk(min) = 57.3088 ms	Vonesa kohore maksimale Vk(max) = 1163.9471 ms
--	---	--

Nga grafiku i dhënë në figurën 40 shihet se nga 200 mesazhet e transmetuara vonesa kohore është e ndryshueshme në mes asaj minimale që është **57ms** deri tek ajo maksimale **1163 ms** ndërsa vonesa kohore mesatare është nën vlerën **100 ms** pra **85 ms**.

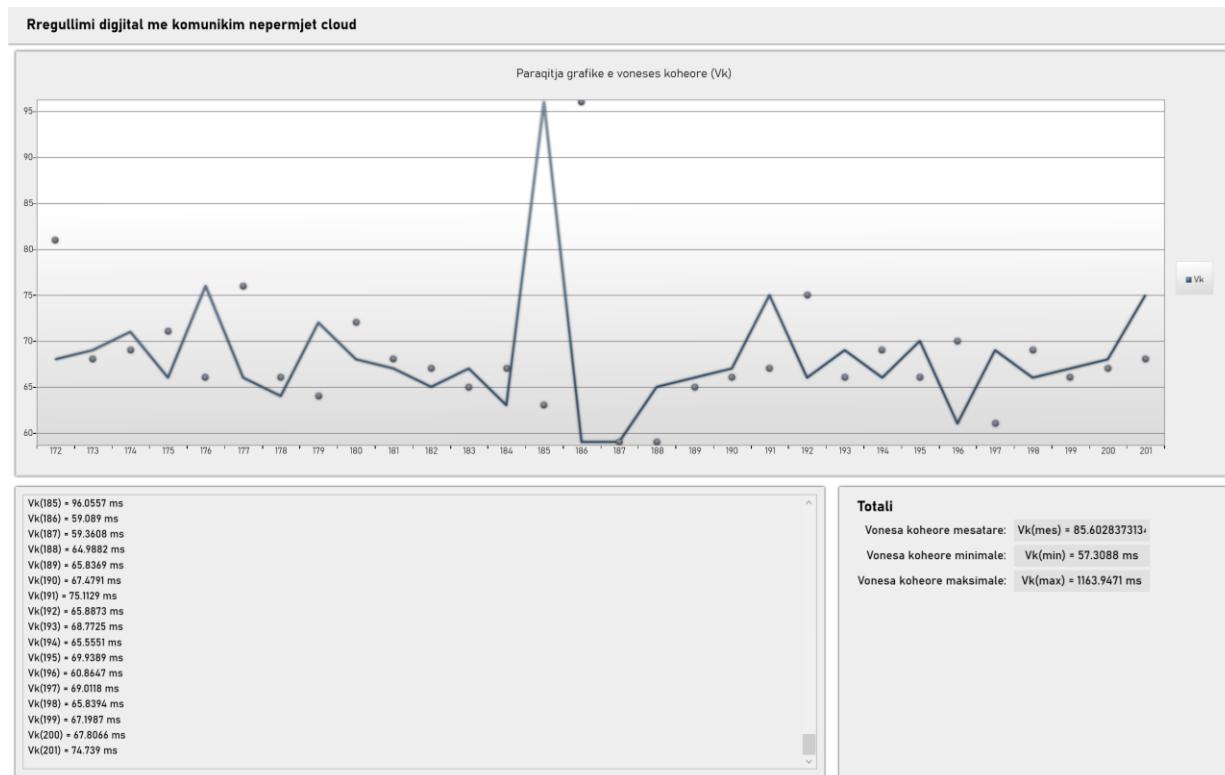


Figura 41: Grafiku i vonesës kohore për 200 matje

5.2.3 Vonesa kohore në 300 matje

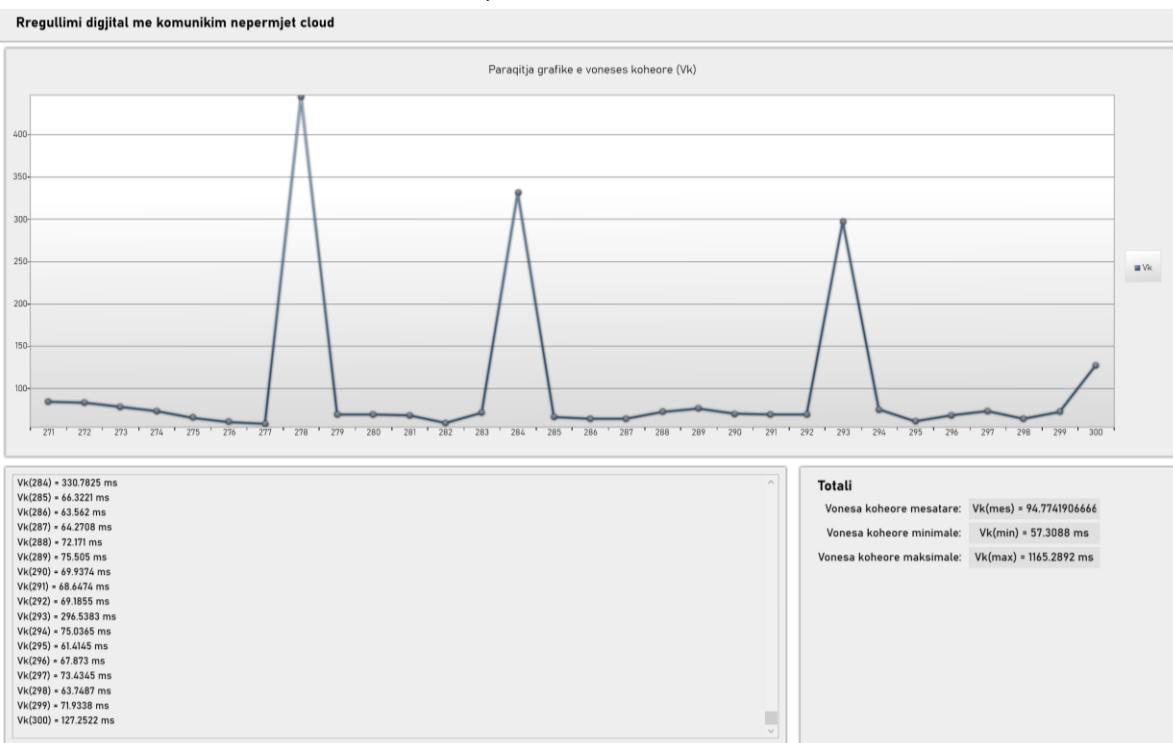


Figura 42: Grafiku i voneses kohore për 300 matje

Vonesa kohore mesatare V _k (mes) = 94.77419 ms	Vonesa kohore minimale V _k (min) = 57.3088 ms	Vonesa kohore maksimale V _k (max) = 1165.2892 ms
--	---	--

Nga grafiku i dhënë në figurën 41 shihet se nga 300 mesazhet e transmetuara vonesa kohore është e ndryshueshme në mes asaj minimale që është **57ms** deri tek ajo maksimale **1165 ms** ndërsa vonesa kohore mesatare është nën vlerën **100 ms** pra **94 ms**.

5.2.4 Vonesa kohore në 400 matje

Vonesa kohore mesatare V _k (mes) = 92.50969 ms	Vonesa kohore minimale V _k (min) = 57.3088 ms	Vonesa kohore maksimale V _k (max) = 1190.4333 ms
--	---	--

Nga grafiku i dhënë në figurën 42 shihet se nga 400 mesazhet e transmetuara vonesa kohore është e ndryshueshme në mes asaj minimale që është **57ms** deri tek ajo maksimale **1190 ms** ndërsa vonesa kohore mesatare është nën vlerën **100 ms** pra **92 ms**.

Analiza e eksperimenteve dhe rezultatet dalëse

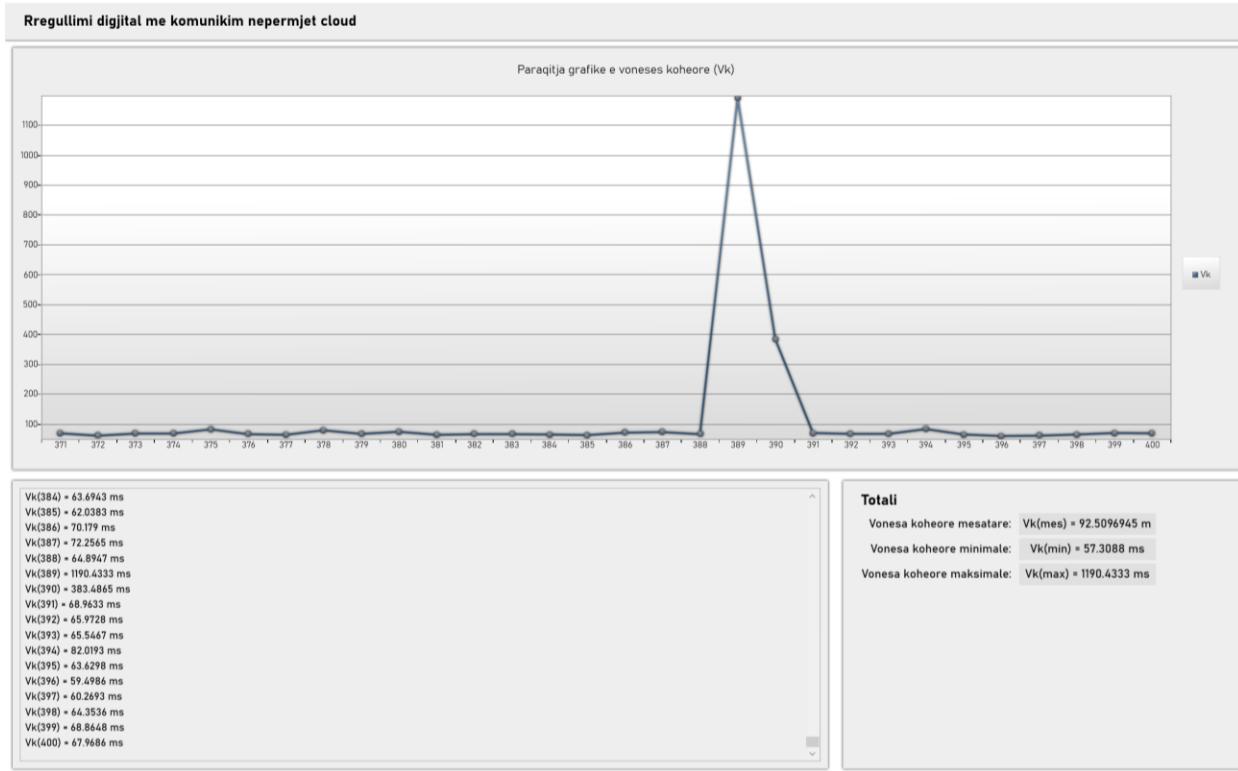


Figura 43: Grafiku i vonesës kohore për 400 matje

5.2.5 Vonesa kohore në 500 matje

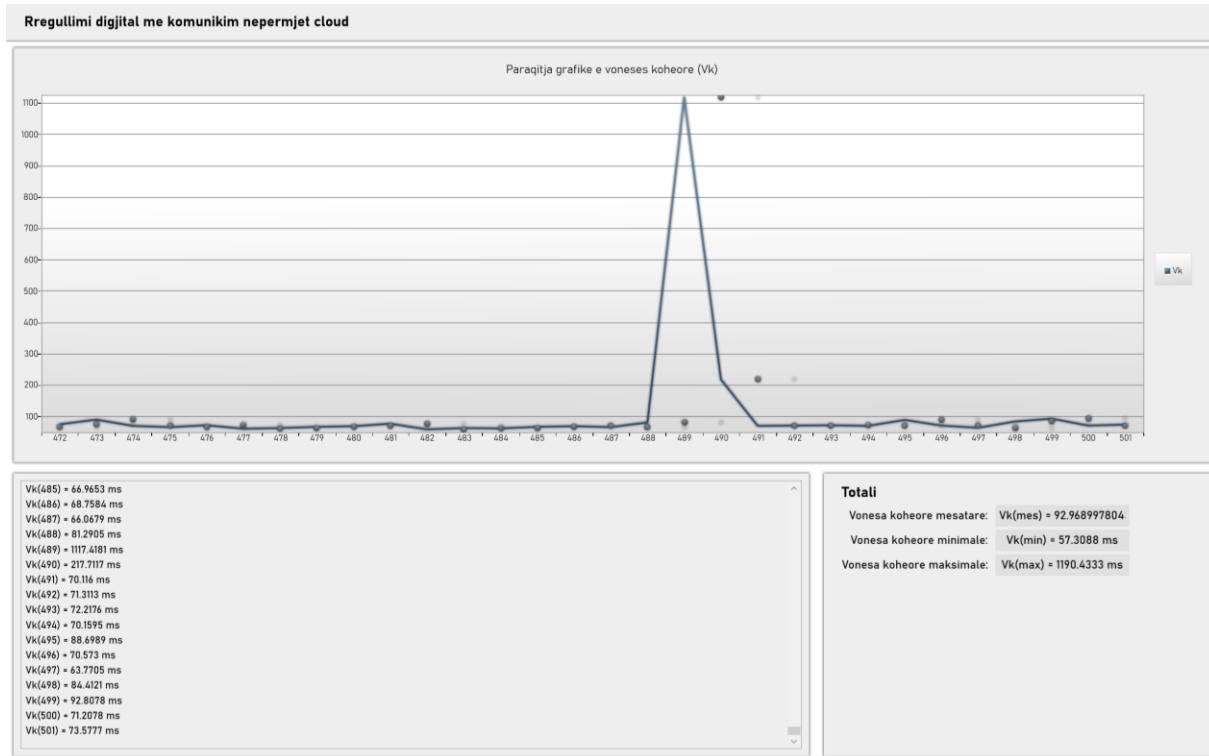


Figura 44: Grafiku i Voneses kohore per 500 matje

Vonesa kohore mesatare $Vk(mes) = 92.96899 \text{ ms}$	Vonesa kohore minimale $Vk(min) = 57.3088 \text{ ms}$	Vonesa kohore maksimale $Vk(max) = 1190.4333 \text{ ms}$
---	--	---

Nga grafiku i dhënë në figurën 43 shihet se nga 500 mesazhet e transmetuara vonesa kohore është e ndryshueshme në mes asaj minimale që është **57ms** deri tek ajo maksimale **1190 ms** ndërsa vonesa koheore mesatare është nën vlerën **100 ms** pra **92 ms**.

5.3 Rezultatet nga matja e vonesës kohore në internet jo-stabil

Si pasoje e jo-stabilitetit të internetit vonesa kohore e mesazheve gjatë transmetimit është e madhe por rregullimi digjital ka ndodhur por me vonesë. Të dhënët që janë transmetuar nga mikrokontrolleri 01 në mikrokontrollerin 02 janë realizuar me suksesë. Rezultatet e 200 matjeve të bëra janë si në vijim.

5.3.1 Vonesa kohore në 100 matje

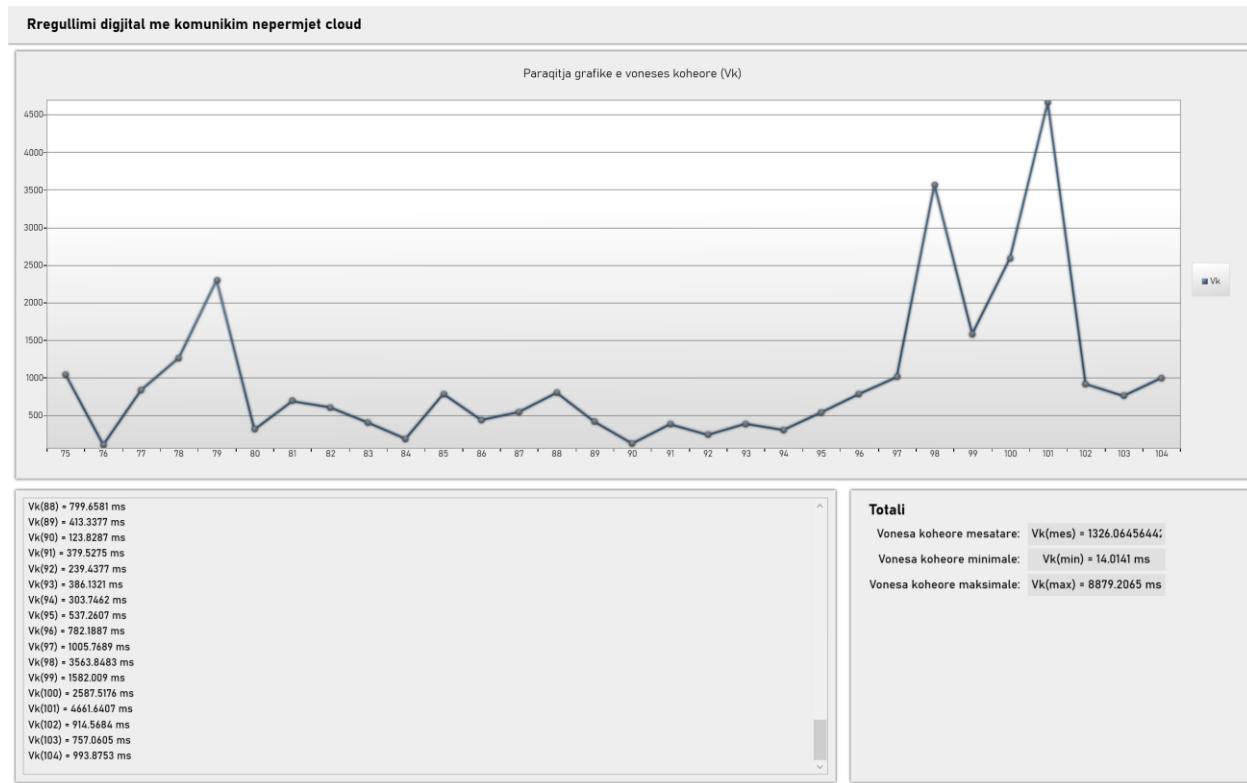


Figura 45: Grafiku i vonesës kohore për 100 matje në internet jo-stabil

Vonesa kohore mesatare Vk(mes) = 1326.0645 ms	Vonesa kohore minimale Vk(min) = 14.0141 ms	Vonesa kohore maksimale Vk(max) = 8879.2065 ms
---	---	--

Nga grafiku i dhënë ne figurën 44 shihet se nga 100 mesazhet e transmetuara vonesa kohore është e ndryshueshme në mes asaj minimale qe është **14 ms** deri tek ajo maksimale **8879 ms** ndërsa vonesa kohore mesatare është shumë mbi vlerën **100 ms** pra **1326 ms**.

5.3.2 Vonesa kohore ne 200 matje

Vonesa kohore mesatare Vk(mes) = 1276.0025 ms	Vonesa kohore minimale Vk(min) = 8.0849 ms	Vonesa kohore maksimale Vk(max) = 12909.4303 ms
---	--	---

Nga grafiku i dhënë ne figurën 45 shihet se nga 200 mesazhet e transmetuara vonesa kohore është e ndryshueshme në mes asaj minimale që është **8 ms** deri tek ajo maksimale **12909 ms** ndërsa vonesa kohore mesatare është shumë mbi vlerën **100 ms** pra **1273 ms**.

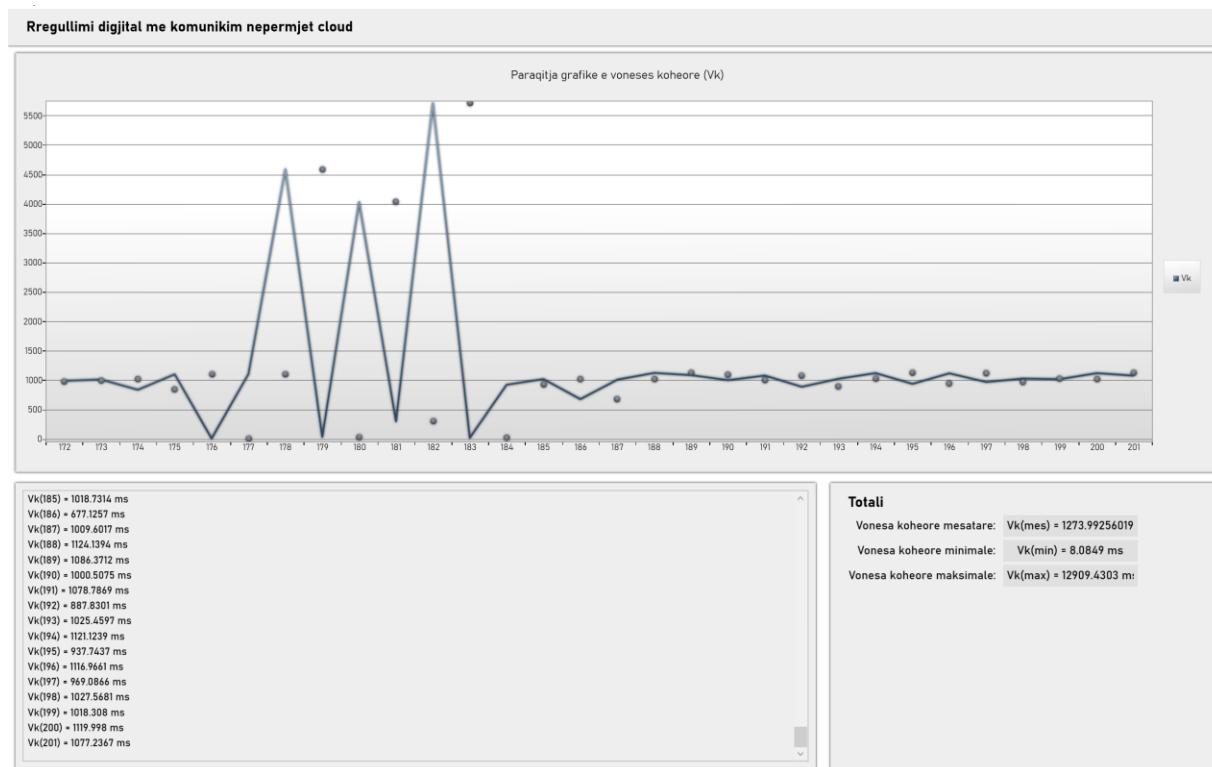


Figura 46: Grafiku i vonesës kohore për 200 matje në internet jo-stabil

5.4 Rezultatet nga eksperimenti Test në mes të mikrokontrollerëve

Eksperimenti i cili i vë në një unazë (*en. loop*) të pa ndërprerë komunikimin e dy mikrokontrollerave i cili është realizuar çdo 10 sekonda, në të cilin është bartë mesazhi i njejtë por duke e rritur variablën Autonumber për 1, ka rezultuar me suksesë vetëm në rastin e testimit të tij me internet stabil. Në momentin kur kishte ramje të internetit (nuk ekzistonte komunikim me cloud), mesazhi humbiste pasi që nuk kishte mundësi të transmetohej deri në shërbimin në cloud, në këtë mënyrë komunikimi në mes mikrokontrollerve nuk mund të vazhdonte.

Në rastin e internetit stabil, vonesa kohore e transmetimit të mesazheve është dhënë në tabelën në vazhdim.

Vonesa kohore mesatare $V_k(\text{mes}) = 10.8513 \text{ ms}$	Vonesa kohore minimale $V_k(\text{min}) = 3.6399 \text{ ms}$	Vonesa kohore maksimale $V_k(\text{max}) = 77.4615 \text{ ms}$
--	---	---

Grafiku i vonesës kohore është dhënë ne figurën ne vazhdim.

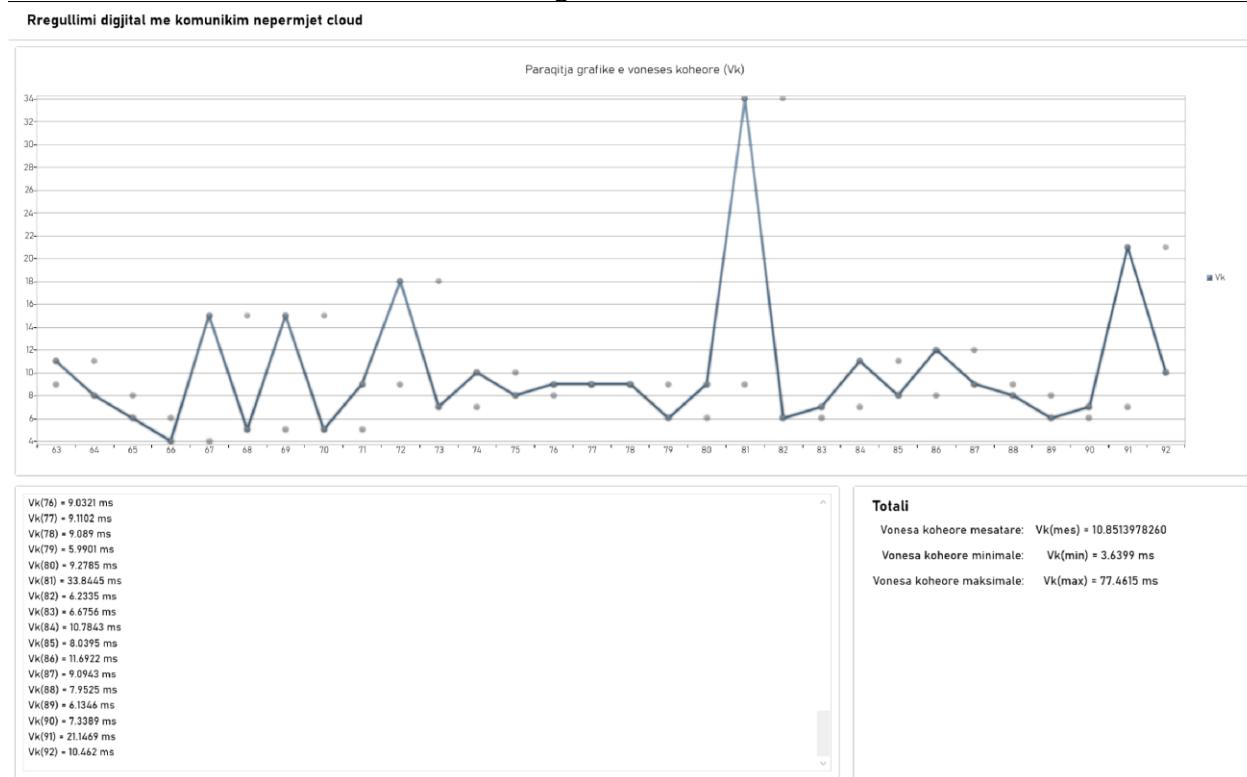


Figura 47: Grafiku i vonesës kohore për 92 matje në internet stabil te eksperimentit Test

Nga grafiku i dhënë ne figurën 46 shihet se nga 92 mesazhet e transmetuara vonesa kohore është e ndryshueshme në mes asaj minimale që është **3 ms** deri tek ajo maksimale **77 ms** ndërsa vonesa kohore mesatare është nën vlerën **100 ms** pra **10 ms**.

Në vazhdim është paraqitur diagrami i rregullimit me komunikim në eksperimentin e dytë.

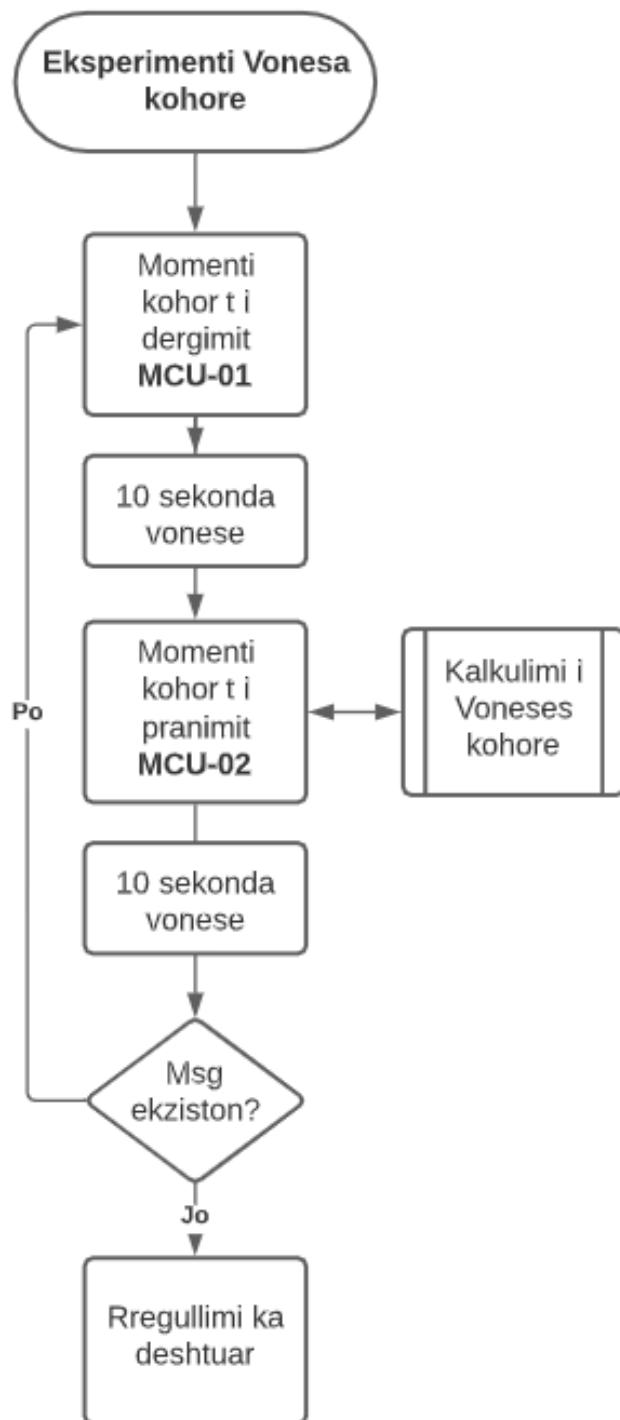


Figura 48: Diagrami për rregullim për eksperimentin Test

6 Diskutime dhe konkluzione

Gjatë punës në këtë temë diplome arritëm që të zhvillojmë një komunikim në mes dy mikrokontrollerve me njeri-tjetrin nëpërmjet platformave cloud. Ashtu siç edhe u analizua më lartë, ne përdorëm Andy X si platformë të shpërndarë për transmetim dhe pamë që këto platforma janë të nevojshme për këto lloje të komunikimit. Me infrastrukturën ekzistuese a mund të ekzistojnë pajisje gjithëmone të ndezura dhe të lidhura? Infrastruktura ekzistuese së bashku me atë që është në zhvillim si 5G dhe platformat cloud me sistemet e shpërndara për transmetim do të mundësojnë që pajisjet elektrike të mund të jenë të lidhura në internet dhe të komunikojnë me njëra-tjetrën.

Gjatë analizës së rezultateve faktori kyç që duhet ti kushtohet rëndësi është edhe vonesa kohore. Në dy eksperimentet e bëra, të ndalemi tek ai i pari kur **MCU-01** dërgon momentin kohore t të dërgimit të mesazhit në **MCU-02**. Ky i fundit bënë krahasimin e dy momenteve kohore atij të **dërguar** dhe **pranuar**, kjo diferenca paraqet **vonesën kohore**. Sa më vogël që të jetë kjo diferenca themi që aq më i shpejtë dhe i saktë do të jetë rregullimi digjital me komunikim, por vlenë edhe e kundërta. Eksperimenti i parë në dy rastet e lidhjes me internet (kur është stabil dhe jo-stabil), në të dy rastet ky eksperiment u ekzekutua pa-humbje të informatës, por në rastin e internetit jo-stabil kishim vonesa shumë të mëdha ($>1\text{s}$). Pra, nga ky eksperiment mund të përfundojmë që në vendet ku nevojitet saktësi (si fabrika) në pajisje duhet patjetër që të kemi lidhje të pandërprerë dhe stabile.

Sic shihen edhe në diagramin e dhënë më lartë në figurën 39, për rregullim të një sistemi në mes dy MCU-ve për këtë eksperiment kemi rregullim nëse vonesa kohore është më e vogël se 100 ms.

Në rastin e eksperimentit të dytë ku mikrokontrolleret transmetojnë te njëjtin mesazh nga njeri tek tjetri, kur ky eksperiment është ekzekutuar në lidhje stabile ka rezultuar me vonesë kohore mesatare **10.8 ms** dhe është realizuar rregullimi digjital me komunikim. Ashtu sic është treguar ne figurën 48, në rastin kur mesazhi nuk mund të transmetohet komunikimi në mes të dy MCU-ve nuk është i mundur. Në rastin kur ekzekutimi i këtij eksperimenti ka ndodhur në lidhje jo-stabile, komunikimi ka dështuar dhe rregullimi nuk është arritur pasi qe mesazhi nga njeri MCU nuk ka mund të transmetohet deri tek tjetri nëpërmjet shërbimit në cloud.

Nga ky dështim kuptuam që softueri i MCU-së nuk tenton që të dërgoj prap të njëjtin mesazh nëse lidhja nuk ka ekzistuar. Pra, kuptuam që jo të gjitha sistemet janë perfekte dhe si zgjidhje që eksperimenti i dytë të funksionojë me vonesa të mëdha kohore në lidhje jo-stabile është që në rastin kur nuk ekziston lidhja, MCU të tentojë prap që të dërgoj informatën. Ky problem tek platformat e shpërndara për transmetim nuk është i pranishëm pasi që në rastin e mos pranimit të informatës nga aplikacioni, këto platforma provojnë të dërgojnë përsëri deri në n tentime.

7 Bibliografia

Referencat nga literaturat

- [1] DATASHEET – Raspberry Pi Compute Module 3+ 1 January 2019 [retrieved at 05.01.2021].
- [2] Jon Skeet. C# in Depth, 4th Edition. How C# workd and why, 2018.

Referencat nga burimet elektronike

- [3] Azure App Service

<https://azure.microsoft.com/en-us/services/app-service/web> [retrieved at 05.01.2021]

- [4] 5 Ways The Internet Of Things Will Make Marketing Smarter

<https://www.forbes.com/sites/salesforce/2014/08/30/5-ways-iot-marketing-smarter/?sh=737dd2554163> [retrieved at 05.01.2021]

- [5] Visual Studio 2019 Release Notes [Online]. Available at:

<https://docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes> [retrieved at 05.01.2021].

- [6] GitHub [Online]. Available at:

<https://github.com/> [retrieved at 05.01.2021].

- [7] Enes Hoxha. GitHub – mcu-data-streaming-cloud, [Online]. Available at:

<https://github.com/eneshoxha/mcu-data-streaming-cloud> [retrieved at 05.01.2021].

- [8] Raspberry Pi 3 Model B - GPIO. Available at:

<https://2.bp.blogspot.com/-YJUVP6rtWM/W8Uq5AY9Y5I/AAAAAAAADFE/oYqIv84VGX83AkZZg3aCaNwhrbknOSnwCLcBGAs/s1600/raspberry-pi-3-gpio-pin-chart-with-pi.jpg> [retrieved at 05.01.2021].

- [9] AndyX. GitHub – andyxTerminal, [Online]. Available at:

<https://github.com/buildersoftdev/andyxterminal> [retrieved at 05.01.2021].

- [10] AndyX. GitHub – andyx, [Online]. Available at:

Bibliografia

<https://github.com/buildersoftdev/andyx> [retrieved at 05.01.2021].

- [11] An overview of Windows 10 IoT Core – Windows 10 IoT Core. Available at:

<https://docs.microsoft.com/en-us/windows/iot-core/windows-iot-core#:~:text=Windows%2010%20IoT%20Core%20is%20a%20version%20of,of%20more.%20If%20you%27re%20ready%20to%20go%20to> [retrieved at 05.01.2021].