VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
KOMPIUTERIŲ INŽINERIJOS KATEDRA

Lech Gulbinovič

AŠTUONIŲ SKILČIŲ MIKROVALDIKLIŲ PANAUDOJIMO *ETHERNET* TINKLO ĮRENGINIUOSE GALIMYBIŲ TYRIMAS

FEASIBILITY STUDY OF 8-BIT MICROCONTROLLER FOR ETHERNET APPLICATIONS

Baigiamasis magistro darbas

Kompiuterių inžinerijos studijų programa, valstybinis kodas 621H69001
Kompiuterių technologijų specializacija
Elektros ir elektronikos inžinerijos mokslo kryptis

Vilnius, 2010

TURINYS

SANTRUMPOS

ARP – *Address Resolution Protocol* – adreso nustatymo protokolas

CSS – *Cascading Style Sheets* – skriptine stiliaus kalba

DMA – *Direct memory access* – tiesioginis atminties pasiekimas

FIFO – *First In, First Out* – pirmas įėjo, pirmas išėjo

HTML – *HyperText Markup Language* – hiperteksto žymėjimo kalba

HTTP – *Hypertext Transfer Protocol* – hiperteksto perdavimo protokolas

I2C – *Inter-Integrated Circuit* – nuoseklioji duomenų perdavimo sąsaja

ICMP – *Internet Control Message Protocol* – interneto valdymo žinučių protokolas

IDS – *Intrusion Detection System* – įsilaužimo aptikimo sistema

IP – *Internet Protocol* – interneto protokolas

LAN – *Local Area Network* – vietinis tinklas

MTU – *Maximum Transfer Unit* – maksimalus paketo dydis

OS – *Operation System* – operacinė sistema

OSI – *Open Systems Interconnection* – atvirų sistemų sąveika

RAM – *Random-access memory* – atsitiktinio pasiekimo atmintis

RTT – *Round Trip Time* – paketo nuėjimo ir grįžimo laikas

SMTP – *Simple Mail Transfer Protocol* – paprastas pašto perdavimo protokolas

SNMP – *Simple Network Management Protocol* – paprastas tinklo stebėjimo protokolas

SPI – *Serial Peripheral Interface* – nuoseklioji duomenų perdavimo sąsaja

TCL – *Tool Command Language* – skriptinės programavimo kalbos

TCP – *Transport Control Protocol* - transporto lygmens protokolas

TOS – *Type of Service* – serviso tipas

TTL – *Time To Live* – paketo gyvavimo laikas

UDP – *User Datagram Protocol* – transporto lygmens protokolas

USART – *Universal asynchronous receiver/transmitter* – universalus asinchroninis imtuvas-siųstuvas

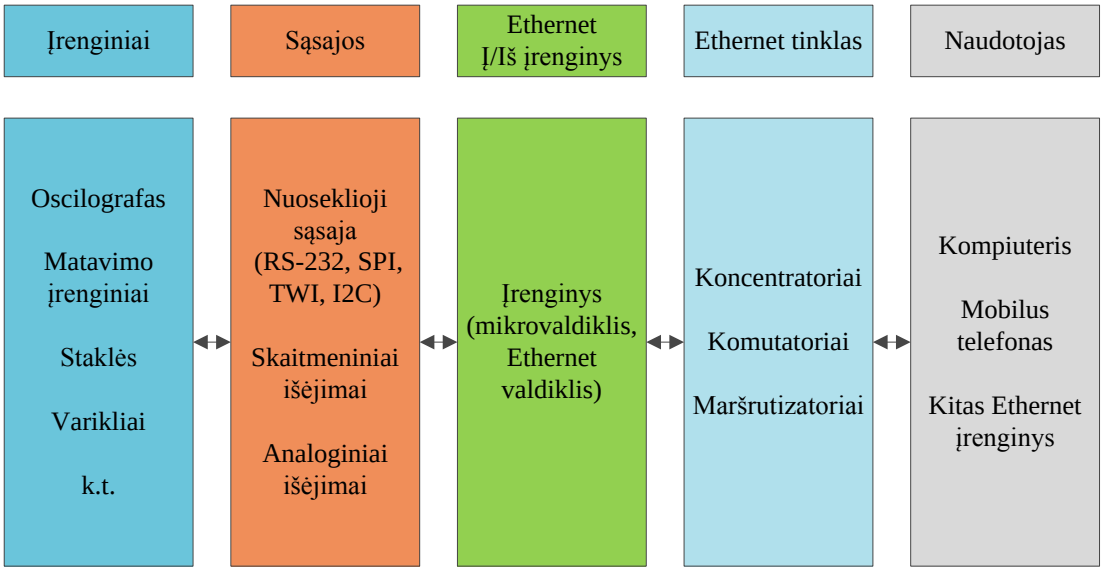
ĮVADAS

XXI amžiuje internetas yra labai lanksti, patogi ir efektyvi duomenų perdavimo priemonė. Beveik visa duomenų perdavimo technologija internetu remiasi *Ethernet* (angl. *Ethernet*) protokolu pagrindų. Kartu su *Ethernet* protokolų tinkle naudojamas TCP/IP protokolų rinkinys užtikrinantis vientisą duomenų perdavimą tinkle. Šiuo metu yra labai daug įrenginių, kurie yra prijungti prie globalaus interneto tinklo. Tai gali būti įvairiausias funkcijas atliekantys kompiuteriai, specialūs įrenginiai, duomenų serveriai ir t.t. Duomenų ir taikomųjų programų naudotojams serverio funkciją gali atlikti ne tik 32/64 skilčių mikroprocesoriai, bet ir daug mažesni ir pigesni 8 arba 16 bitų mikrovaldikliai.

Šio darbo tikslas – tyrimo metodikos sudarymas ir priemonių parinkimas, tiriant tinklo įrenginius, kurių pagrindą sudaro 8 skilčių mikrovaldikliai. Tokių įrenginių duomenų apdorojimo sparta nėra didelė, todėl svarbu nustatyti kokio tipo uždavinius galima spręsti panaudojant 8 skilčių mikrovaldiklius *Ethernet* tinkluose, ištirti jų charakteristikas. Išnagrinėti mikrovaldiklių naudojimo *Ethernet* tinkluose galimybes ir veikimo spartos didinimo galimybes.

1. ANALOGIŠKŲ ĮRENGINIŲ APŽVALGA

Šiame skyriuje apžvelgsime analogiškus įrenginius, kurių funkcija – įrenginių valdymas nuotolinių būdu per kompiuterių tinklą. Tokie įrenginiai gali būti panaudoti kitų įrenginių valdymui, *Ethernet* sąsajos konvertavimui į RS-232, I2C sąsajas, jutiklių duomenų parodymų persiuntimui. Dažniausiai įrenginys valdomas per interneto naršyklę, sąsajos konvertavimo funkcijos panaudojimas reikalauja papildomos programos kompiuteryje. *Ethernet* Į/Iš įrenginių veikimo schema parodyta (1.1 pav.).



1.1 pav. Ethernet Į/Iš įrenginių veikimo schema

Tokio tipo įrenginius gamina: *FIVECO*, *KORENIX*, *WINFORD*, *MOXA*, *PERLE*, *ADVANTECH*, *HW-Group*, ir kiti. Trumpai apžvelgsime tokius įrenginius.

1.1. *Fiveco* įrenginių apžvalga

FMOD-TCP yra *Fiveco* gamintojo įrenginių valdymo įrenginys skirtas valdymui nuotolinių būdu per *Ethernet* tinklą [18]. Leidžia valdyti pramonines bei integruotas sistemas, turi galimybę prijungti temperatūros, slėgio, judesio ir kitus jutiklius. Galimybę nuotoliniu būdu stebėti bei valdyti prievadus, jutiklių parodymus panaudojus TCP/IP protokolą. Naudojami protokolai ICMP, ARP, IP, UDP, TCP, HTTP.

Įrenginys turi tokias sąsajas kaip UART, I2C, Į/Iš prievadus, 10 skilčių analoginius/skaitmeninius įėjimus. Visi įėjimo bei išėjimo signalai yra skaitmenizuojami ir perduodami internetu panaudojus perdavimo protokolus TCP/UDP.

FMOD-TCP įrenginys yra gaminamas kaip atskira plokštė be korpuso, yra patogus integravimui į kitą sistemą. Įrenginio vaizdas parodytas (1.2 pav.).

1.2 pav. Įrenginių valdymo įrenginys *FMOD-TCP*

Fiveco dar gamina tokius įrenginių valdymo įrenginius kaip *Ethernet Motion control device 48 V 10 A* naudojamas šepetinių ir be šepečių variklių iki 480 W valdymui su integruotu reguliatoriumi, kuris leidžia reguliuoti variklio sukimosi greitį. Valdymas atliekamas – specialia programa (1.3 pav.).

1.3 pav. Įrenginių valdymo įrenginys
Ethernet Motion control device 48 V 10 A

FMOD-LEDSEQUENCER – tai apšvietimo valdymo įrenginys. Leidžia valdyti apšvietimą, kurti apšvietimo animacija. Visas valdymas atliekamas per *Ethernet* tinklą, galimybė nustatyti lempučių mirksėjimo seką ir periodą (1.4 pav.).

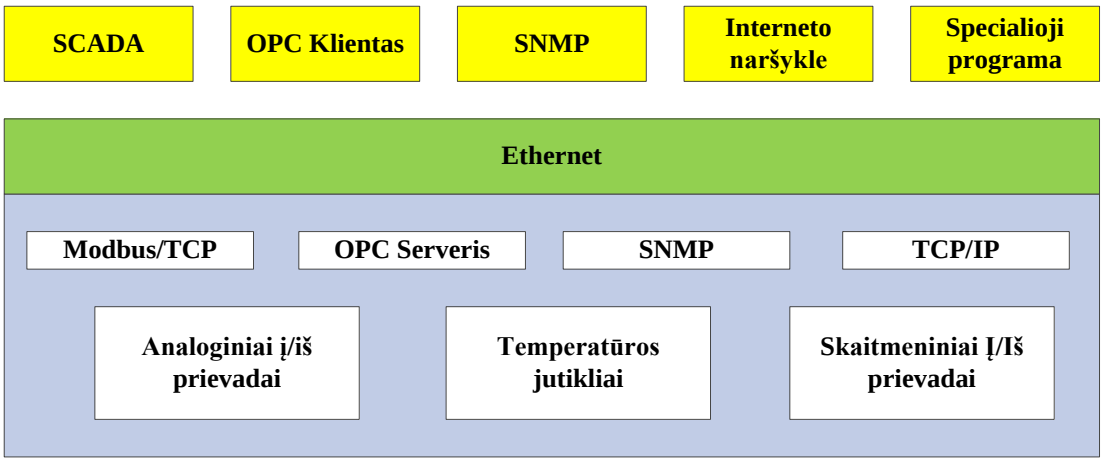
1.4 pav. Įrenginių valdymo įrenginys
FMOD-LEDSEQUENCER

Apžvelgus *Fiveco* gamintojo įrenginius galima pasakyti, kad yra gaminami trijų rūšių įrenginiai: universalus, variklių valdymui, apšvietimo valdymui.

1.2. *Korenix* įrenginių apžvalga

Korenix gamintojas siūlo tokius produktus kaip: *JetIO*, *JetPort*. Visi šie produktai yra skirti įrenginiams valdyti per *Ethernet* tinklą [19].

JetIO – tai serija išmaniųjų Į/Iš serverių, naudojamų stebėjimui bei valdymui. Įrenginiai gaminami su *Ethernet* prieiga ir daugybė prievadų, tokių kaip: analoginiai/skaitmeniniai įėjimai, skaitmeniniai įėjimo/išėjimo prievadai, temperatūros jutiklių prievadai. Naudotojas gali lengvai nuskaityti arba išvesti duomenis į prievadus.



1.5 pav. *JetIO Ethernet* Į/Iš serverio architektūra

Paveiksle (1.5 pav.) pateikta JetIO Ethernet Į/Iš serverio architektūra. Žemesnis lygmuo vaizduoja fizinį serverio lygmenį, tai yra Į/Iš prievadai. Vidinis lygmuo parodo valdymo

sąsaja per *Ethernet* tinklą. Aukščiausias lygmuo yra taikomųjų programų lygmuo, per kurį yra valdomi bei stebimi fiziniai serverio prievadai.

JetIO serijos įrenginiai skiriasi tarpusavyje prievadų skaičiumi. Galimi 4, 8 ir 14 įvesties prievadų modeliai bei 8 išvesties prievadus turintys moduliai. Įrenginio pavyzdys parodytas (1.6 pav.).

1.6 pav. *JetIO 6550 Ethernet* Į/Iš serveris

JetPort – tai nuoseklių prievadų serveris, tai yra išmoningasis RS-232/422/485 sąsajų į *Ethernet* sąsają keitiklis. *JetPort* įrenginiai leidžia prijungti prie nuosekliųjų sąsajų tokius įrenginius kaip kortelių skaitytuvą, matavimo įrenginius, terminalus ir viskas tai bus pasiekiama per *Ethernet* tinklą.

1.7 pav. *JetPort 5604 Ethernet to RS-232* serveris

JetPort serijos įrenginiai skiriasi tarpusavyje nuosekliųjų prievadų skaičiumi ir tipu. Įrenginio pavyzdys parodytas (1.7 pav.).

1.3. *Moxa* įrenginių apžvalga

Moxa gamintojas siūlo tokius nuotolinio valdymo Į/Iš įrenginius: *Active Ethernet I/O, Peer to Peer I/O, Serial Remote I/O, Modular Remote I/O* [20].

Active Ethernet I/O serijos įrenginiai naudojami valdyti Į/Iš prievadus per *Ethernet* tinklą. Skiriasi tarpusavyje Į/Iš prievadų skaičiumi.

Peer to Peer I/O serijos įrenginiai yra skirti perduoti analoginį signalą iš vieno taško į kitą. Signalo lygio perdavimui yra naudojamas *Ethernet* tinklas.

Serial Remote I/O serijos įrenginiai yra skirti valdyti įrenginius su RS-485 sąsaja per *Ethernet* tinklą. Papildomai turi valdomus Į/Iš prievadus.

Modular Remote I/O serijos įrenginiai yra universalūs nuotolinio valdymo serveriai, gali turėti neribotą skaičių Į/Iš prievadų, RS-232/485 sąsajų. Išplėsti šio serverio galimybėms yra naudojami specialūs moduliai. *Moxa* gaminamo įrenginio pavyzdys parodytas (1.8 pav.).

1.8 pav. *Moduliar Remote I/O ioLogik E4200* įrenginys

Apžvelgus *Moxa* gamintojo siūlomus įrenginius galime pasakyti, kad yra platus įrenginių pasirinkimas. Gamintojas *Moxa* išsiskiria nuo visų kitų gamintojų įrenginių valdymo ir stebėjimo galimybėmis. Įrenginius galima valdyti per *Ethernet* tinklą, bet duomenis apie valdomų įrenginių būseną galima gauti SMS žinute, el. laišku, SNMP žinutėmis.

Apibendrinant analogiškų įrenginių apžvalgą įrenginius galime suskirstyti į tokias grupes pagal paskirtį:

- Universalieji, skirti valdyti nuotoliniu būdu Į/Iš prievadus;
- Keitikliai tokie kaip, RS-232, RS-422, RS-485, I2C į *Ethernet* sąsaja;
- Specialūs, skirti tam tikrai funkcijai įvykdyti (variklio valdymas, apšvietimo valdymas).

Visi gamintojai savo gaminius yra suskirstę į panašias paskirties grupes. Skirtingi gamintojai siūlo savo valdymo taikomąsias programas, bet beveik visi gamintojai siūlo ir valdymą per interneto naršyklę. Reikalinga tik interneto naršyklė, nereikalauja papildomos programinės įrangos įdiegimo ir visi įrenginiai yra laisvai pasiekiami iš bet kurio pasaulio taško.

2. ĮRENGINIO PROJEKTAVIMAS

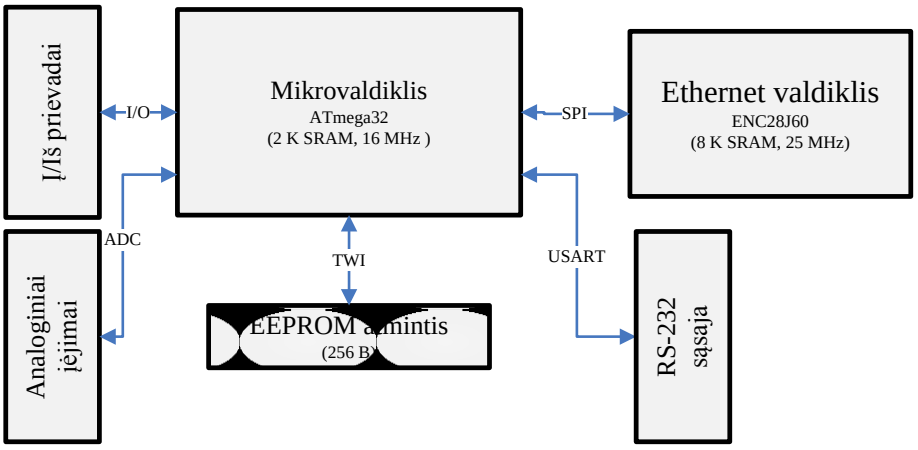
Aštuonių skilčių mikrovaldiklių panaudojimo *Ethernet* tinkle galimybių tyrimui atlikti reikia suprojektuoti įrenginį, skirtą kitų įrenginiams valdyti. Tas įrenginys bus valdomas per interneto naršyklę. Įrenginio projektavimo tikslas geriau suprasti kaip tokio tipo įrenginiai veikia. Taip pat įrenginio projektavimas leis lengvai keisti programos kodą, aparatinę dalį, įrenginio funkcijų rinkinį, darbo režimus.

Projektuojamo įrenginio pagrindas bus mikrovaldiklis *ATmega32* [23]. Mikrovaldiklis turi 32 K programos atmintį ir 2 K operatyviąją atmintinę, gali dirbti 16 MIPS greičiu. *Ethernet* technologija bus realizuota naudojant *Ethernet* valdiklį *ENC28J60* [21], kuris užtikrina 10 Mbps perdavimo standartą. Mikrovaldiklis galės apdoroti 1518 baitų ilgio paketus, to užtenka, norint persiųsti paprastą HTML (angl. *HyperText Markup Language*) puslapį. Visas įrenginys bus tiesiogiai prijungtas prie interneto tinklo ir jame turi būti realizuoti šie protokolai: IP, TCP, UDP, ICMP, ARP ir HTTP.

Įrenginys turės atlikti tokias funkcijas kaip:

- *Ethernet* paketų priėmimas ir išsiuntimas;
- Atsakyti į ARP užklausas;
- Atsakyti į ICMP (*echo request*) užklausas;
- Priimti ir išsiųsti TCP bei UDP paketus;
- Atlikti žiniatinklio serverio funkciją;

Įrenginį sudarys mikrovaldiklis, *Ethernet* valdiklis ir Į/Iš prievadai. Įrenginio blokinė schema parodyta (2.1 pav.). Mikrovaldiklis atliks viso įrenginio valdymo funkciją ir žiniatinklio serverio funkciją, *Ethernet* valdiklis užtikrins *Ethernet* sąsaja, Į/Iš prievadus galima bus valdyti nuotoliniu būdu prisijungus prie serverio žiniatinklio.



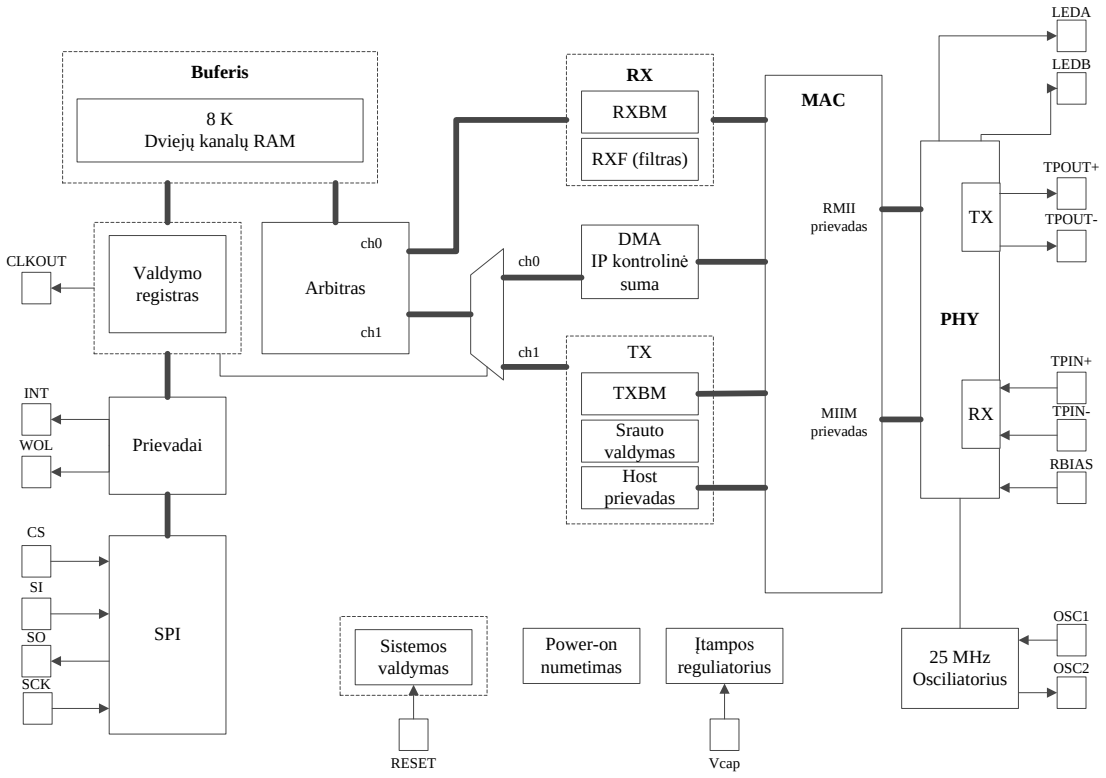
2.1 pav. Įrenginio blokinė schema

Programa mikrovaldikliui bus rašoma *C* programavimo kalbą panaudojant atviro kodo bibliotekas tokias kaip *WINAVR* ir *AVRLIB* [9,10].

Projektuojamas įrenginys turės atlikti kitų įrenginių valdymo funkciją nuotoliniu būdu prisijungiant prie įrenginio per internetą, iš mobiliojo telefono arba asmeninio kompiuterio. Toks įrenginys galės būti pritaikytas, valdant intelektinį namą, nutolusius objektus ir pramonėje.

2.1. *ENC28J60 Ethernet* valdiklio galimybių apžvalga

Mikrovaldiklio sąsaja su *Ethernet* tinklu bus realizuota *Ethernet* valdikliu *ENC28J60* [21]. Apžvelgsime valdiklio galimybes. *Ethernet* valdiklis *ENC28J60* prijungiamas prie mikrovaldiklio per SPI sąsaja, kuri palaiko iki 10 Mbps duomenų perdavimo spartą, pats valdiklis palaiko *Ethernet* 10BASET standartą. Valdiklis atitinka visas IEEE 802.3 specifikacijas. Valdiklyje yra realizuoti du žemiausi tinklo modelių lygmenys, tai yra fizinis (PHY) ir kanalinis (MAC). Turi galimybę filtruoti ateinančius paketus tam, kad neapkrautų mikrovaldiklio nereikalingų paketų apdorojimu. Supaprastinta *ENC28J60* valdiklio blokinė schema parodyta (2.2 pav.).

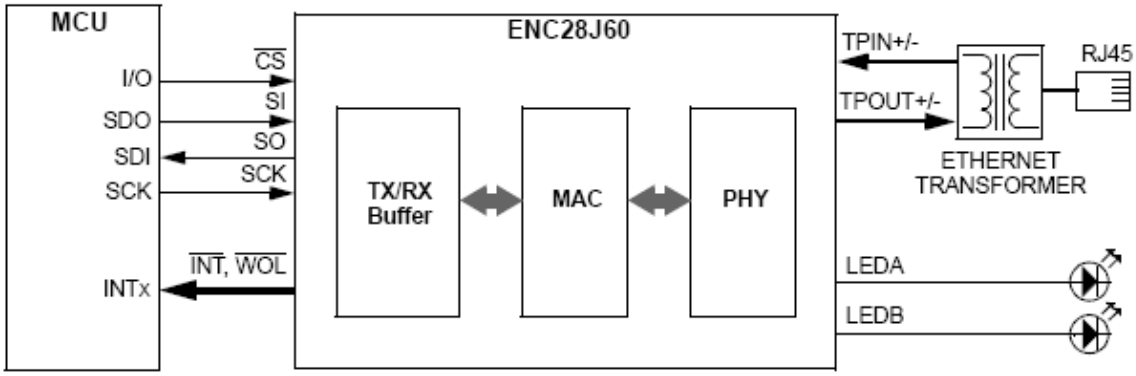


2.2 pav. *ENC28J60 Ethernet* valdiklio blokinė schema

Valdiklį sudaro septyni pagrindiniai blokai:

1. SPI sąsaja, leidžianti prijungti *ENC28J60* valdiklį prie mikrovaldiklio;
2. Valdymo registrai leidžia valdyti ir stebėti valdiklio darbą;
3. Dviejų kanalų RAM buferis skirtas priiminėti bei išsiųsti paketus;
4. Arbitras valdo priėjimą prie RAM buferio;
5. Magistralės sąsaja interpretuoja atėjusius duomenis iš SPI sąsajos: ar tai komandos, ar tai duomenys;
6. Ryšio lygmenį (MAC) realizuojantis blokas;
7. Fizinio lygmens blokas kuris koduoja/dekoduoja analoginį signalą ateinantį per vytą porą iš *Ethernet* tinklo.

Tipinė taikymo grandinė parodyta (2.3 pav.). *ENC28J60* valdiklio prijungimui prie *Ethernet* tinklo reikia dviejų skiriamųjų transformatorių ir kelių pasyviųjų elementų.



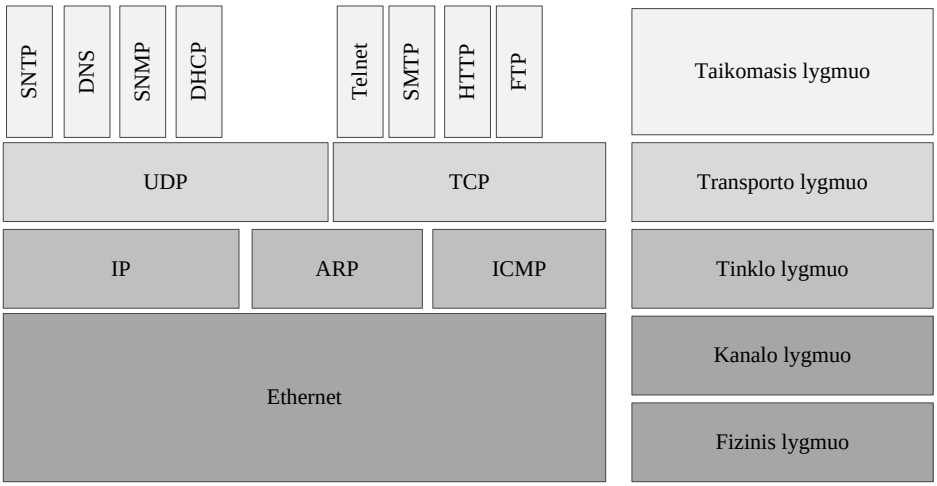
2.3 pav. *ENC28J60 Ethernet* valdiklio tipinė jungimo schema

Detaliau ištirsime valdiklio galimybes priimant ir apdorojant paketus. Siekiant optimizuoti paketų apdorojimo procesą valdiklyje yra numatyta galimybė filtruoti atėjusius paketus ir nereikalingus atmesti, pagal tokius požymius:

- *Unicast* paketai – filtravimas pagal gavėjo MAC adresą;
- *Pattern Match* – filtravimas pagal tam tikrą 64 bitų požymį;
- *Magic* paketai – filtravimas pagal specialų paketą;
- *Hash* lentelė – filtravimas pagal gavėjo MAC adreso lentelę;
- *Multicast* paketai – filtravimas pagal *multicast* paketus;
- *Broadcast* paketai – filtravimas pagal *broadcast* paketus.

Yra galimybė aktyvuoti daugiau negu vieną filtrą.

Kaip jau buvo minėta, *Ethernet* valdiklis realizuoja du žemiausius tinklo lygmenis – fizinį ir ryšio. Aukštesni tinklo lygmenys realizuojami programiškai mikrovaldikliu. Tinklo lygmenys ir naudojami protokolai parodyti (2.4 pav.).



2.4 pav. Tinklo lygmenys ir protokolai

Apžvelgus ir ištyrus *Ethernet* valdiklio galimybes galima padaryti tokias išvadas:

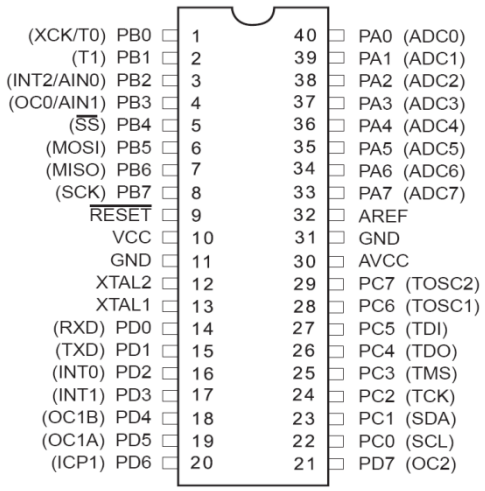
- *ENC28J60 Ethernet* valdiklis yra lengvai valdomas ir kontroliuojamas per SPI sąsają;
- *Ethernet* valdiklyje visiškai realizuoti fizinis ir ryšio tinklo lygmenys, bei valdiklis atitinka visas IEEE 802.3 specifikacijas;
- *Ethernet* valdiklis palaiko 10BASET standartą;
- Papildomai valdiklyje yra realizuotos paketų filtravimo galimybės, tai leidžia mažiau apkrauti mikrovaldiklį nereikalingais paketais.

2.2. Pagrindinės mikrovaldiklio funkcijos ir jų programinis realizavimas

Visą įrenginio valdymą atliks mikrovaldiklis [4]. Mikrovaldiklio programos funkcijas galima išskaidyti į dalis taip:

- Mikrovaldiklio inicializacija ir prievadų paskirties konfigūravimas;
- *Ethernet* valdiklio inicializacija;
- Išorinės EEPROM atminties inicializacija;
- ASK inicializacija;
- UART inicializacija;
- Įvesties/Išvesties prievadų valdymas ir stebėjimas;
- Temperatūros jutiklių valdymas ir parodymų nuskaitymas;
- Statistikos rašymas į EEPROM atmintį;
- Laikmačio inicializacija;
- TCP/IP stekas;
- Žiniatinklio puslapio generavimas;
- Paketų priėmimas ir apdorojimas;
- Terminalo RS232 per TCP realizavimas;
- Terminalo RS232 per UDP realizavimas;

Mikrovaldiklio inicializacija pradedama nuo prievadų konfigūravimo. Mikrovaldiklio prievadai bus skirti temperatūros jutikliams, įvesties/išvesties prievadams, EEPROM atminčiai, *Ethernet* valdikliui, RS232 prievadui, maitinimui, kvarciniam rezonatoriui. Mikrovaldiklio prievadų paskirtis parodyta (2.5 pav.).



2.5 pav. *ATmega32* išvadų išdėstymas ir pavadinimai

Mikrovaldiklio prievadų paskirtis parodyta 2.1 lentelėje.

2.1 lentelė. Mikrovaldiklio ATmega32 išvadų paskirtis

Išvadas	**Kojos numeris**	**Tipas**	**Paskirtis**
Vcc	10	įvestis	+5V maitinimas
GND	11, 31	įvestis	Maitinimo žemė
PA0	40	įvestis	Temperatūros jutiklis 1
PA1	39	įvestis	Temperatūros jutiklis 2
PA2	38	įvestis	Temperatūros jutiklis 3
PA3	37	įvestis	Temperatūros jutiklis 4
PA4	36	įvestis	Įvesties prievadas 1
PA5	35	įvestis	Įvesties prievadas 2
PA6	34	įvestis	Įvesties prievadas 3
PA7	33	įvestis	Įvesties prievadas 4
PB4	5	išvestis	SPI sąsajos SS prievadas
PB5	6	išvestis	SPI sąsajos MOSI prievadas
PB6	7	įvestis	SPI sąsajos MISO prievadas
PB7	8	išvestis	SPI sąsajos SCK prievadas
PC0	22	Išvestis/įvestis	EEPROM Duomenų prievadas
PC1	23	Išvestis	EEPROM SCL prievadas
PC2	24	išvestis	Išvesties prievadas 1
PC3	25	išvestis	Išvesties prievadas 2
PC4	26	išvestis	Išvesties prievadas 3
PC5	27	išvestis	Išvesties prievadas 4
PC6	28	išvestis	Išvesties prievadas 5
PC7	29	išvestis	Išvesties prievadas 6
PD0	14	įvestis	RS232 RX prievadas
PD1	15	išvestis	RS232 TX prievadas
XTAL1	13	įvestis	Kvarcinis rezonatorius
XTAL2	12	įvestis	Kvarcinis rezonatorius

Į/Iš prievadų būsenos nustatymas vykdomas per specialiuosius registrus, o programos kodas atrodo taip:

```
/* PA4 prievadas nustatomas kaip įėjimas */
DDRA&= ~(1<<PA4);
/* Nustatomas loginis lygis „0” */
PORTA&= ~(1<<PA4);

/* PC2 prievadas nustatomas kaip išėjimas */
DDRC|= (1<<DDC2);
/* Nustatomas loginis lygis „0” */
PORTC&= ~(1<<PC2);
```

Per registrą DDRx nustatoma prievado kryptis įėjimas/išėjimas, PORTx registras nustato loginį lygį prievado išėjime.

Įvesties/išvesties prievadų valdymas ir stebėjimas yra atliekamas per naudotojo grafinę aplinką. Įvesties prievadai yra skirti loginiam lygiui nustatyti 0 V ar 5 V. Išvesties prievaduose atvirkščiai galima nustatyti loginius lygius 0 V arba 5 V.

Įvesties prievado būsenos nustatymo pavyzdys parodytas programos kode:

```
//PORTA.4
if (PINA & (1<<PA4))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTA.4 [<img src=\"gre\">]<br>"));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTA.4 [<img src=\"red\">]<br>"));}
```

Jeigu prievado įėjime loginis „1“ tai naudotojas mato žalią mygtuką, jeigu loginis lygis „0“ tai rodomas raudonas mygtukas.

Loginio lygio išvedimas į prievadą parodytas programos kode:

```
//PORTC.2
if (PORTC & (1<<PC2))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.2 <img src=\"gre\" align=middle>
[ "));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.2 <img src=\"red\" align=middle>
[ "));}

plen=fill_tcp_data_p(buf,plen,PSTR("<a href=\"21\">on</a> | "));
plen=fill_tcp_data_p(buf,plen,PSTR("<a href=\"20\">off</a> ]<br>"));
```

Prievado loginio lygio būseną parodoma tai pat kaip ir įvesties prievado būseną. Jeigu prievado įėjime loginis „1“ tai naudotojas mato žalią mygtuką, jeigu loginis lygis „0“ tai rodomas raudonas mygtukas. Prievado loginio lygio būsenos nustatymas vykdomas siunčiant mikrovaldikliui užklausą nuorodos pavidalu pvz.:

http://192.168.1.100/21

http://192.168.1.100/20

Siunčiant kodą ***21*** bus nustatytas prievado PC2 loginis lygis „1“, siunčiant kodą ***20*** bus nustatytas loginis lygis „0“. Atsiųstos nuorodos apdoroja mikrovaldiklis. Nuorodos atpažinimo ir apdorojimo kodas:

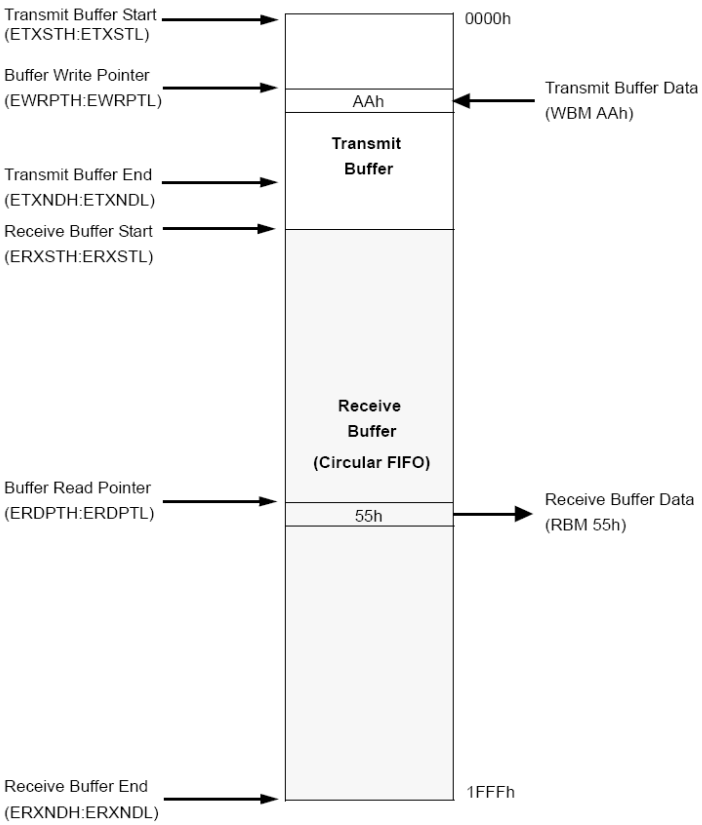
```
//PORTC.2 on
if (strncmp("21",(char *)&(buf[dat_p+5]),2)==0)
{
PORTC|= (1<<PC2);     // PC2 nustatomas loginis „1”
plen=print_webpage_out(buf);
goto SENDTCP;
}

//PORTC.2 off
if (strncmp("20",(char *)&(buf[dat_p+5]),2)==0)
{
PORTC &= ~(1<<PC2);   // PC2 nustatomas loginis „0”
plen=print_webpage_out(buf);
goto SENDTCP;
}
```

Programos kodas tikrina atėjusią nuorodą ir atitinkamai nustato prievadų būseną.

Ethernet valdiklio inicializavimui nustatomas SPI (angl. *Serial Peripheral Interface*) prievadas MASTER režimu ir nustatomas jo taktinis dažnis [21]. *Ethernet* valdiklis paleidžiamas. Nustatomas *Ethernet* valdiklio buferio atminties padalinimas į dvi dalis. Priėmimo ir persiuntimo atmintis, kur bus saugomi paketai. Buferis yra 8 K atminties. Priėmimo buferio ribos yra nustatomos per valdymo registrus ERXSTH:ERXSTL ir ERXNDH:ERXNDL. Priėmimo buferis užsipildo automatiškai naudoja DMA (angl. *Direct Memory Access*) atėjus duomenų paketams ir dirba FIFO režimu. DMA yra automatinis rašymas ir skaitymas iš atminties, valdiklis automatiškai rašo į atmintį atėjusius paketus ir skaičiuoja kontrolines sumas. Buferio atminties paskirstymas parodytas (2.6 pav.). Nustatomas išsiuntimo paketo maksimalus dydis 1518 B.

Ethernet valdiklio inicializacijai reikalingi tokie duomenis kaip įrenginio MAC adresas, IP adresas, buferio dydis. Visi kintamieji persiunčiami *Ethernet* valdiklio inicializavimo funkcijoms, programos kodas:

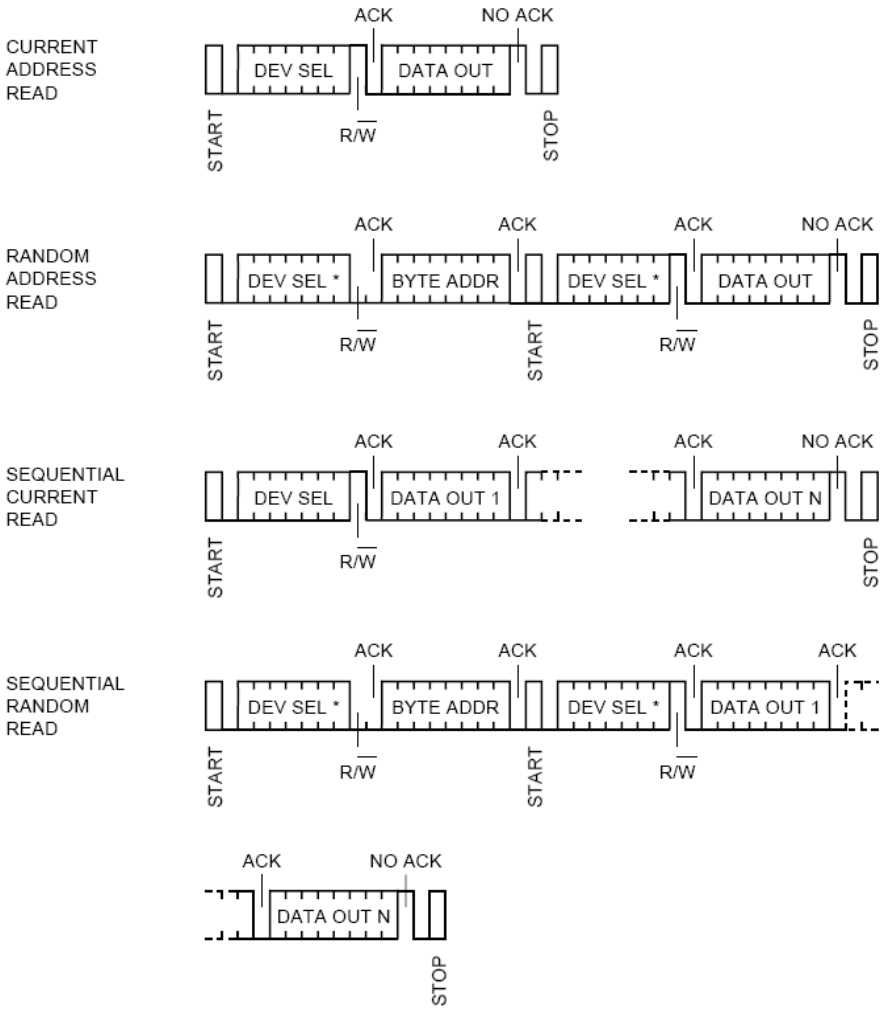
```
//Nustatomas įrenginio MAC adresas
static uint8_t mymac[6] = {0x54,0x55,0x58,0x10,0x00,0x25};
//Nustatomas įrenginio IP adresas
static uint8_t myip[4] = {192,168,20,60};
//Buferio dydžio nustatymas
#define BUFFER_SIZE 1518
static uint8_t buf[BUFFER_SIZE+1];
/*inicializacija enc28j60*/
enc28j60Init(mymac);
```



2.6 pav. *Ethernet* valdiklio buferio atminties paskirstymas

Toliau nustatomi *Ethernet* valdiklio paketų filtravimo nustatymai. Naudojami tam, kad mažiau apkrautume valdiklį nereikalingais paketais. Nustatomas leidimas *broadcast* paketams tik ARP protokolui. Visi kiti *unicast* paketai yra atmetami ir priimami tik mums skirti paketai pagal MAC adresą. Nustatomas *Ethernet* valdiklio fizinis 48 skilčių MAC adresas. Nustatomas leidimas paketų priėmimui.

Išorinės EEPROM atminties inicializacija vykdoma iš pradžių nustačius TWI (angl. *Two Wire Interface*) mikrovaldiklio prievadą [24]. Dviejų laidų duomenų perdavimo prievadas turi SCL ir SDA išvadas, pirmasis skirtas taktiniam dažniui generuoti prievadas, kitas naudojamas duomenų persiuntimui. TWI prievado taktinis dažnis nustatomas 100 kHz. Į išorinę EEPROM atmintį galima rašyti ir skaityti duomenis. Skaitymas atliekamas tokiais žingsniais, nustatomas įrenginio prijungto prie TWI sąsajos adresas. Mūsų atveju EEPROM atminties adresas 0xA. Toliau siunčiamas norimo baito adresas. Duomenų skaitymo diagrama parodyta (2.7 pav.).

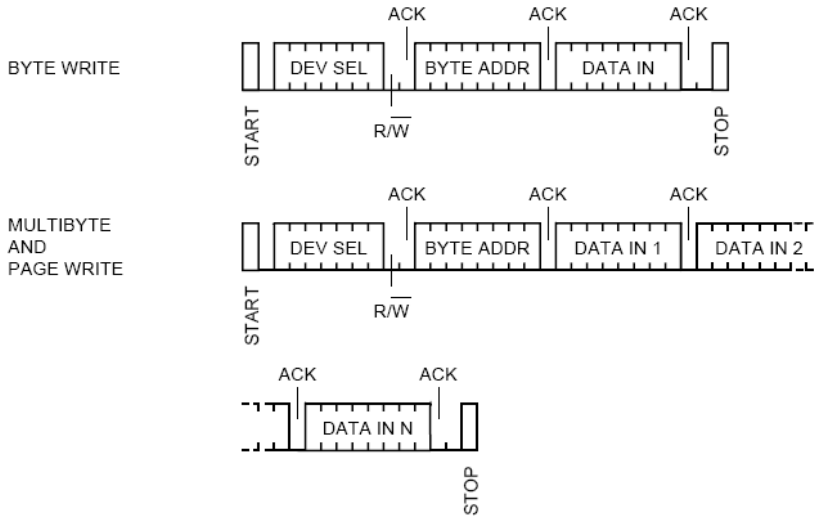


2.7 pav. Skaitymo iš EEPROM atminties diagramos

Rašymas į atmintį atliekamas tokiais žingsniais, nustatomas įrenginio prijungto prie TWI sąsajos adresas. Mūsų atveju EEPROM atminties adresas 0xA. Toliau siunčiamas norimo baito adresas ir duomenų baitas. Duomenų rašymo diagrama parodyta (2.8 pav.).

EEPROM atminties inicializacijos kodas:

```
char eeprom_buffer;   //rezervuota atmintis
char eeprom_buffer2;  //rezervuota atmintis

//EEPROM inicializacija
eeprom_init();

//rašymas į EEPROM atmintį
eeprom_write_byte(0x00, eeprom_buffer);

//Skaitymas iš EEPROM atminties
eeprom_read_byte(0x00, &eeprom_buffer2);
```



2.8 pav. Rašymo į EEPROM atmintį diagramos

ASK (Analoginis skaitmeninis keitiklis) inicializuojamas nustačius atraminės įtampos dydį nuo 0 V iki 2,56 V [23]. Nustatomas dažnio daliklis. Panaudojami mikrovaldiklio vidiniai 10 skilčių ASK. ASK įvesties prievado nustatymas atliekamas per ADMUX registrą. Skaitmenizuotą analoginio signalo reikšmė yra išsaugoma ADCH:ADCL registruose. Prie atitinkamų AS keitiklio prievadų yra prijungti temperatūros jutikliai. Temperatūros jutiklių parodymus galima pamatyti per naudotojo grafinę aplinką. Taip pat yra nuimami temperatūros parodymai paskutiniu 6 valandų kas 4 minutes ir įrašomi į EEPROM atmintį.

ASK inicijacijos ir analoginės įtampos nuskaitymo programos kodas:

```
//ASK inicializacija
void a2dInit_my(void)
{
          ADCSRA |= (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
          ADMUX|= (1<<REFS1)|(1<<REFS0);//|(1<<ADLAR);
}


// Nustatomas ASK prievadas
void a2dSetChannel_my(unsigned char ch)
{
          ADMUX      &=0b11111000;
          ADMUX |=ch;
}
//Nuskaitomi ASK parodymai
uint16_t a2dConvert10bit_my(unsigned char ch)
{
          ADMUX &=0b11111000;
          ADMUX |=ch;
          ADCSRA |= (1<<ADSC);
while( bit_is_set(ADCSRA, ADSC) );
// Laukia kol bus pabaigtas nuskaitymas
          return ((ADCL)|((ADCH)<<8)); //Grąžinama 10 skilčių reikšme
}
```

USART (angl. *Universal Sinchronous Asinchronous Receiver Transmiter*) inicializuojamas nustačius RX ir TX prievadus mikrovaldiklyje [23]. Yra nustatomi tokie RS232 sąsajos parametrai kaip:

- Duomenų perdavimo greitis: 9600 bps;
- Duomenų formatas: 8 bitų;
- Lyginumo bitai: nėra;
- Stop bitas: 1;
- Srauto valdymas: nėra.

USART inicializavimo, siuntimo ir skaitymo programos kodas:

```
void USART_init(void) {
           // Nustatomas USART duomenų perdavimo greitis
           UBRRH = (uint8_t)(UART_BAUD_CALC(UART_BAUD_RATE,F_OSC)>>8);
           UBRRL = (uint8_t)UART_BAUD_CALC(UART_BAUD_RATE,F_OSC);
           // Leidžiamas siuntimas ir priemimas
           UCSRB = (1<<RXEN)|(1<<TXEN)|(1<<RXCIE);
           //Asinchronimis režimas 8N1
           UCSRC = (1<<URSEL)|(3<<UCSZ0);
}

//Simbolio siuntimas
void usart_putc(unsigned char c) {
           // Laukia kada bus laisvaas USART
           while(!(UCSRA & (1 << UDRE)));
           UDR = c;              //Siunčia simbolį
}

//Simbolių sekos siuntimas
void uart_puts (char *s) {
           //siunčia simbolių seka kol *s != NULL
           while (*s) {
                      usart_putc(*s);
                      s++;
           }
}
```

USART duomenų priėmimas vykdomas per išorinę pertrauktį:

```
SIGNAL (SIG_UART_RECV)
{ // USART RX interrupt
           unsigned char c;
           c = UDR;
           usart_putc(c);
}
```

Viena iš pagrindinių mikrovaldiklio funkcijų yra programinis TCP/IP steko realizavimas [10]. TCP/IP yra vienas iš populiariausių protokolų rinkinių duomenų perdavimo

komunikacijoje. TCP/IP yra naudojamas web aplikacijoms persiuntinėti, e-mail persiuntimui, duomenų persiuntimui per internetą. Projekte bus naudojamas ne visas TCP/IP stekas, o tik jo dalis. Bus naudojami tik tokie protokolai kaip: ARP, IP, ICMP, UDP, TCP, HTTP.

TCP/IP stekas inicializuojamas viena funkcija:

```
//TCP/IP steko inicializavimas
init_ip_arp_udp_tcp(mymac,myip,mywwwport);
```

Grafinės naudotojo aplinkos žiniatinklio puslapis yra generuojamas tuo metu, kai ateina užklausa šiam puslapiui atvaizduoti. Generuojant puslapį yra surenkami duomenys apie prievadų būsenas, temperatūros parodymus. Generuojamas TCP paketas ir išsiunčiamas naudotojui.

Pagrindinio naudotojo grafinės aplinkos puslapio programos kodas atrodo taip:

```
uint16_t print_webpage_index(uint8_t *buf)
{uint16_t plen;
plen=fill_tcp_data_p(buf,0,PSTR("HTTP/1.0 200 OK\r\nContent-Type:
text/html\r\n\r\n"));
plen=fill_tcp_data_p(buf,plen,PSTR("<html>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<head>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<link rel=\"stylesheet\"
type=\"text/css\" href=\"css.css\">"));
plen=fill_tcp_data_p(buf,plen,PSTR("<title>avrETH</title>"));
plen=fill_tcp_data_p(buf,plen,PSTR("</head>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<body>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<center>"));
//main index
plen=fill_tcp_data_p(buf,plen,PSTR("<br><div class=\"box\">"));
//title
plen=fill_tcp_data_p(buf,plen,PSTR("<div class=\"bt\">"));
plen=fill_tcp_data_p(buf,plen,PSTR("avrETH"));
plen=fill_tcp_data_p(buf,plen,PSTR("</div>"));
//menu
plen=fill_tcp_data_p(buf,plen,PSTR("<div class=\"bm\">"));
plen=fill_tcp_data_p(buf,plen,PSTR("<br><a href=\"men\"
class=\"n\">home</a><br>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<a href=\"out\"
class=\"n\">output</a><br>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<a href=\"inp\"
class=\"n\">input</a><br>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<a href=\"tem\"
class=\"n\">temp</a><br>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<a href=\"ter\"
class=\"n\">terminal</a><br>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<a href=\"abo\"
class=\"n\">about</a><br></div>"));
//source 1
plen=fill_tcp_data_p(buf,plen,PSTR("<div class=\"bs\">"));
plen=fill_tcp_data_p(buf,plen,PSTR("<br>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<u>Output:</u><br><br>"));
//PORTC.2
if (PORTC & (1<<PC2))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.2 [<img src=\"gre\">]<br>"));}
else
```

```
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.2 [<img src=\"red\">]<br>"));}
//PORTC.3
if (PORTC & (1<<PC3))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.3 [<img src=\"gre\">]<br>"));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.3 [<img src=\"red\">]<br>"));}
//PORTC.4
if (PORTC & (1<<PC4))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.4 [<img src=\"gre\">]<br>"));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.4 [<img src=\"red\">]<br>"));}
//PORTC.5
if (PORTC & (1<<PC5))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.5 [<img src=\"gre\">]<br>"));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.5 [<img src=\"red\">]<br>"));}
//PORTC.6
if (PORTC & (1<<PC6))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.6 [<img src=\"gre\">]<br>"));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.6 [<img src=\"red\">]<br>"));}
//PORTC.7
if (PORTC & (1<<PC7))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.7 [<img
src=\"gre\">]<br></div>"));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTC.7 [<img
src=\"red\">]<br></div>"));}
//source 2
plen=fill_tcp_data_p(buf,plen,PSTR("<div class=\"bs\">"));
plen=fill_tcp_data_p(buf,plen,PSTR("<br>"));
plen=fill_tcp_data_p(buf,plen,PSTR("<u>Input:</u><br><br>"));
//PORTA.4
if (PINA & (1<<PA4))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTA.4 [<img src=\"gre\">]<br>"));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTA.4 [<img src=\"red\">]<br>"));}
//PORTA.5
if (PINA & (1<<PA5))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTA.5 [<img src=\"gre\">]<br>"));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTA.5 [<img src=\"red\">]<br>"));}
//PORTA.6
if (PINA & (1<<PA6))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTA.6 [<img src=\"gre\">]<br>"));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTA.6 [<img src=\"red\">]<br>"));}
//PORTA.7
if (PINA & (1<<PA7))
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTA.7 [<img
src=\"gre\">]<br></div>"));}
else
{plen=fill_tcp_data_p(buf,plen,PSTR("PORTA.7 [<img
src=\"red\">]<br></div>"));}
//source 3
plen=fill_tcp_data_p(buf,plen,PSTR("<div class=\"bs\">"));
plen=fill_tcp_data_p(buf,plen,PSTR("<br><u>Temperatura:</u><br>"));
//temperature
//0
plen=fill_tcp_data_p(buf,plen,PSTR("<br>+"));
adcresult=a2dConvert10bit_my(0x00);
HEXtoABC(adcresult);
```

```
plen=fill_tcp_data_p_temp(buf,plen);
plen=fill_tcp_data_p(buf,plen,PSTR("°C  ADC0"));
//1
plen=fill_tcp_data_p(buf,plen,PSTR("<br>+"));
adcresult=a2dConvert10bit_my(0x01);
HEXtoABC(adcresult);
plen=fill_tcp_data_p_temp(buf,plen);
plen=fill_tcp_data_p(buf,plen,PSTR("°C  ADC1"));
//2
plen=fill_tcp_data_p(buf,plen,PSTR("<br>+"));
adcresult=a2dConvert10bit_my(0x02);
HEXtoABC(adcresult);
plen=fill_tcp_data_p_temp(buf,plen);
plen=fill_tcp_data_p(buf,plen,PSTR("°C  ADC2"));
//3
plen=fill_tcp_data_p(buf,plen,PSTR("<br>+"));
adcresult=a2dConvert10bit_my(0x03);
HEXtoABC(adcresult);
plen=fill_tcp_data_p_temp(buf,plen);
plen=fill_tcp_data_p(buf,plen,PSTR("°C  ADC3"));
plen=fill_tcp_data_p(buf,plen,PSTR("</div>"));
plen=fill_tcp_data_p(buf,plen,PSTR("</div>"));
plen=fill_tcp_data_p(buf,plen,PSTR("</center>"));
plen=fill_tcp_data_p(buf,plen,PSTR("</body>"));
plen=fill_tcp_data_p(buf,plen,PSTR("</html>"));
return(plen);
}
```

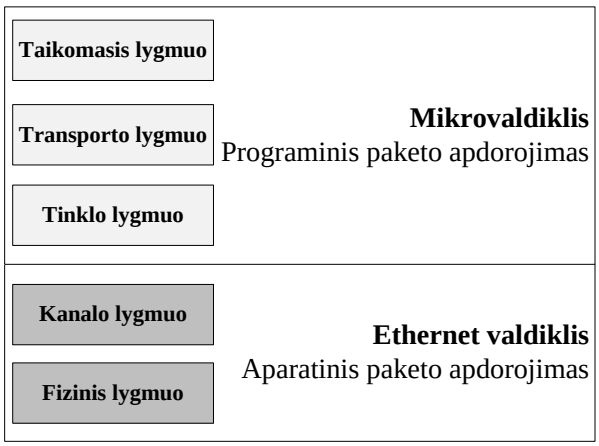
Pagrindinio puslapio HTML kodas atrodo taip:

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="css.css">
<title>avrETH</title>
</head>
<body>
<center>
<br><div class="box">
<div class="bt"> avrETH</div>
<div class="bm">
<br><a href="men" class="n">home</a><br>
<a href="out" class="n">output</a><br>
<a href="inp" class="n">input</a><br>
<a href="tem" class="n">temp</a><br>
<a href="ter" class="n">terminal</a><br>
<a href="abo" class="n">about</a><br></div>
<div class="bs">
<br>
<u>Output:</u><br><br>
PORTC.2 [<img src="gre">]<br>
PORTC.3 [<img src="gre">]<br>
PORTC.4 [<img src="gre">]<br>
PORTC.5 [<img src="gre">]<br>
PORTC.6 [<img src="gre">]<br>
PORTC.7 [<img src="gre">]<br></div>
<div class="bs">
<br>
<u>Input:</u><br><br>
PORTA.4 [<img src="gre">]<br>
PORTA.5 [<img src="gre">]<br>
PORTA.6 [<img src="gre">]<br>
```

```
PORTA.7 [<img src="gre">]<br></div>
<div class="bs">
<br><u>Temperatura:</u><br>
<br>+00,0°C  ADC0
<br>+00,0°C  ADC1
<br>+00,0°C  ADC2
<br>+00,0°C  ADC3
</div>
</div>
</center>
</body>
</html>
```

HTML puslapio fragmento kodai yra saugomi mikrovaldiklio duomenų atmintyje, kiti dinaminiai parodymai yra kuriami realiame laike [25]. Puslapio formavimas vyksta mikrovaldiklio operatyviojoje atmintyje. Kopijuojami HTML kodo fragmentai iš duomenų atminties į operatyviąją atmintį, taip pat realiame laike yra pridedami temperatūros bei prievadų būsenų parodymai.

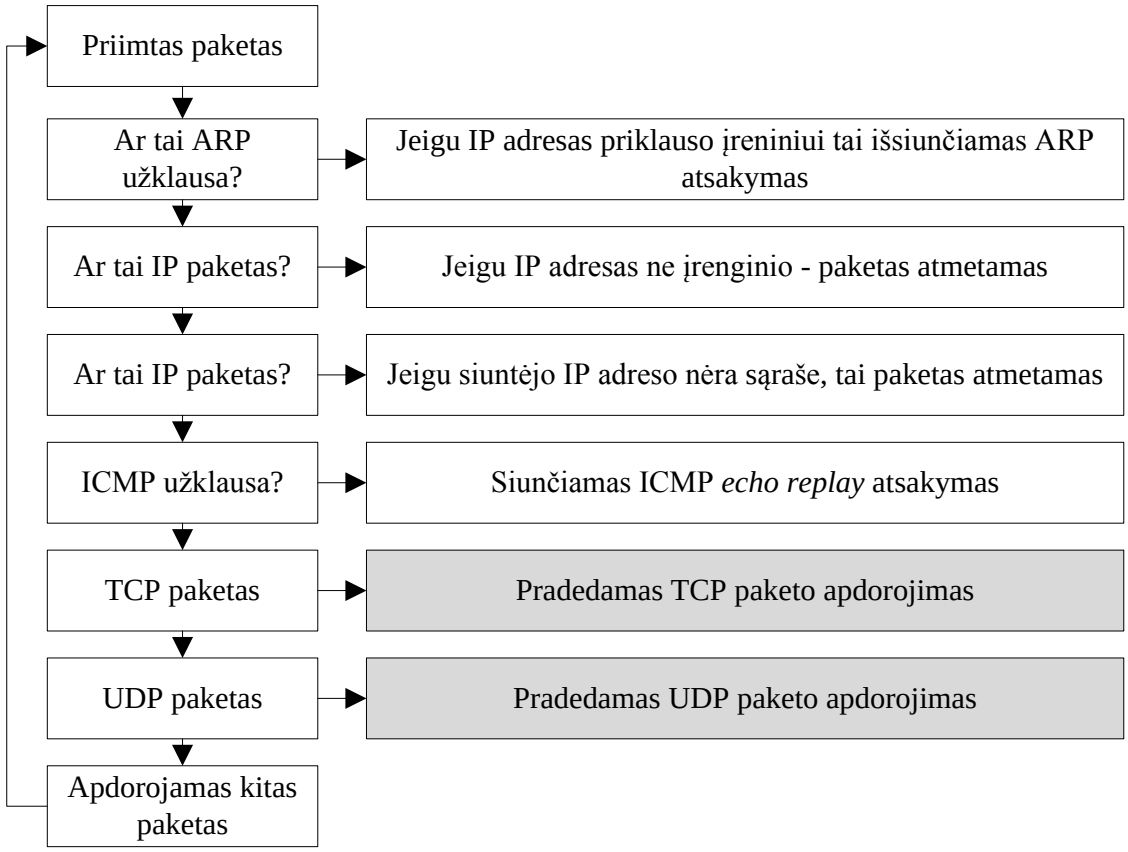
Ethernet paketų apdorojimo procesą galima padalinti į du etapus. Tai fizinio ir kanalinio OSI lygmens apdorojimas, kuris vyksta *Ethernet* valdiklyje, kur fizinis signalas yra keičiamas skaitmeninių ir formuojamas paketas. Kanaliniame lygmenyje yra tikrinama paketo kontrolinė suma ir klaidos. Visa tai realizuojama aparatiškai. Antras paketo apdorojimo etapas tai programinis paketo apdorojimas, kur yra apdorojami tinklo, transporto ir taikomasis OSI lygmuo. Paketo apdorojimo paskirstymas tarp *Ethernet* valdiklio ir mikrovaldiklio parodytas (2.9 pav.).



2.9 pav. Įrenginio OSI lygmenų apdorojimas

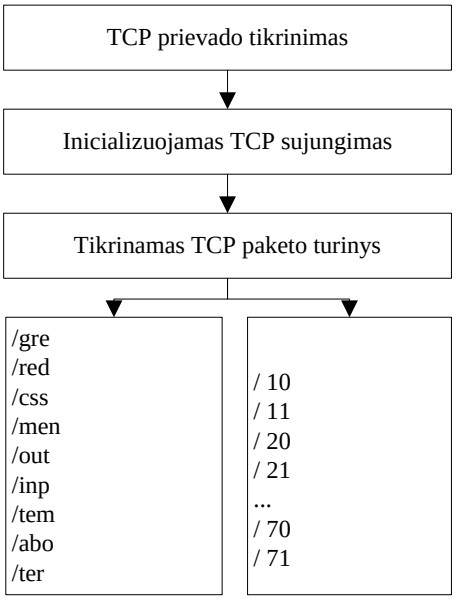
Programinis paketų apdorojimas yra parodytas diagramoje (2.10 pav.). Paketo apdorojimas prasideda nuo to, kad paketas iš *Ethernet* valdiklio buferio persiunčiamas į mikrovaldiklio RAM atminį. Paketas apdorojamas priklausomai nuo protokolo, yra tikrinami IP adresai, MAC adresai, ICMP protokolas, ARP protokolas. Jeigu atėjęs paketas yra TCP, tai

yra inicializuojamas TCP sujungimas, o po to yra tikrinamas TCP paketo turinys. Jeigu atėjęs paketas yra UDP, tai yra tikrinamas UDP prievadas.



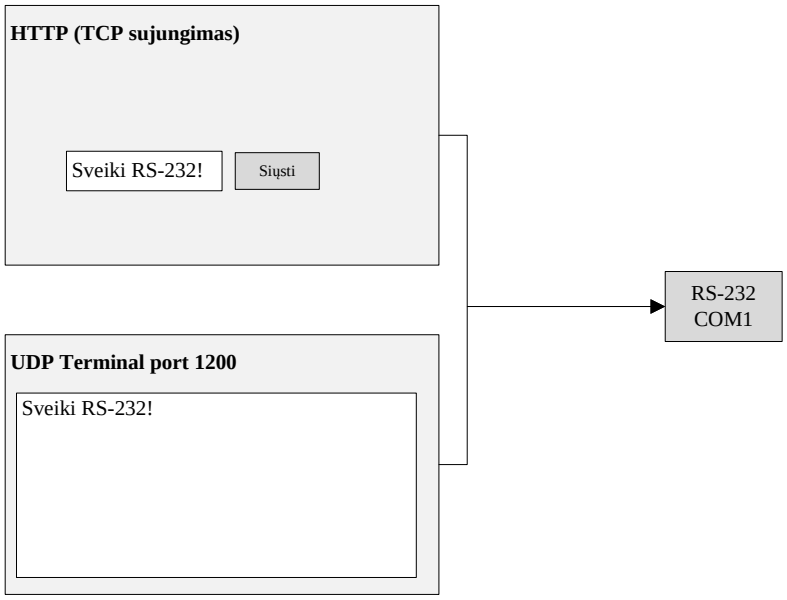
2.10 pav. Programinio paketų apdorojimo diagrama

TCP paketo apdorojimo diagrama parodyta (2.11 pav.). Iš pradžių tikrinamas TCP prievadas. Jeigu tai HTTP protokolui skirtas paketas, tai prievadas 80. Jeigu prievadas lygus 80 tai yra užmezgama TCP sesija ir tikrinamas TCP paketo turinys. Priklausomai nuo paketo turinio yra atliekamos funkcijos, tai gali būti puslapio užklausymas, prievadų valdymas.



2.11 pav. TCP paketo apdorojimo diagrama

Terminalas RS-232 yra realizuotas dviem būdais per TCP ir UDP sujungimą. TCP sujungimas garantuoja, kad paketas pasieks gavėją. TCP sujungimas su RS-232 sąsaja realizuotas per grafinę naudotojo sąsają ir yra pasiekiamas per naudotojo naršyklę. UDP sujungimas su RS-232 yra realizuotas per UDP prievadą ir kaip pavyzdys paimtas 1200 prievadas. UDP paketų pasiekimas nėra garantuotas. UDP sujungimas yra realizuotas per specialia programą vadinamą „*UDP Terminal*“. RS-232 sąsajos valdymas per *Ethernet* schemą yra parodytas (2.12 pav.).



2.12 pav. RS-232 sąsajos valdymas

Apibendrinant visas funkcijas kurias tūrės atlikti mikrovaldiklis, mikrovaldiklio pajėgumo 16 MIPS bus pakankamai apdorojant *Ethernet* paketus, valdant prievadus, matuojant ir renkant temperatūros statistiką.

2.3. Grafinės naudotojo aplinkos projektavimas

Tam, kad būtų galima patogiai ir aiškiai valdyti mikrovaldiklio prievadus ir atvaizduoti prievadų bei mikrovaldiklio būseną reikalinga sąsaja su vartotojų. Geriausiai suprantama ir lengvai valdoma yra grafinė vartotojo sąsaja. Kuriant vartotojo sąsaja bus atsižvelgiama į mikrovaldiklio galimybes. Grafinė vartotojo sąsaja bus realizuota žiniatinklio serverio pagrindu. Mikrovaldiklis atliks žiniatinklio serverio funkcijas, priiminės ir apdoros užklausas.

Visas grafinės aplinkos atvaizdavimas ir apdorojimas bus atliekamas naudotojo kompiuteryje. Mikrovaldikliui nereikės apdoroti grafikos, jis tiktai persiųs duomenis į kompiuterį, o jau kompiuterio naršyklė apdoros grafinę ir tekstinę informaciją. Mikrovaldiklis tik persiuntinės ir priiminės paketus. Grafinė aplinka bus HTML puslapio pavidalo. Mikrovaldiklis generuos HTML puslapį ir formuos paketus iki 1518 B dydžio, kurie bus persiunčiami kompiuteriui. Kompiuterio naršyklėje puslapis bus apdorotas ir atvaizduotas naudotojui kaip grafinė sąsaja.

Tam, kad vartotojo grafinė aplinka būtų lengvai ir aiškiai suprantama kartu su HTML bus naudojamas CSS (angl. *Cascading Style Sheets*) skriptas naudojamas dokumento formavimui [25, 26].

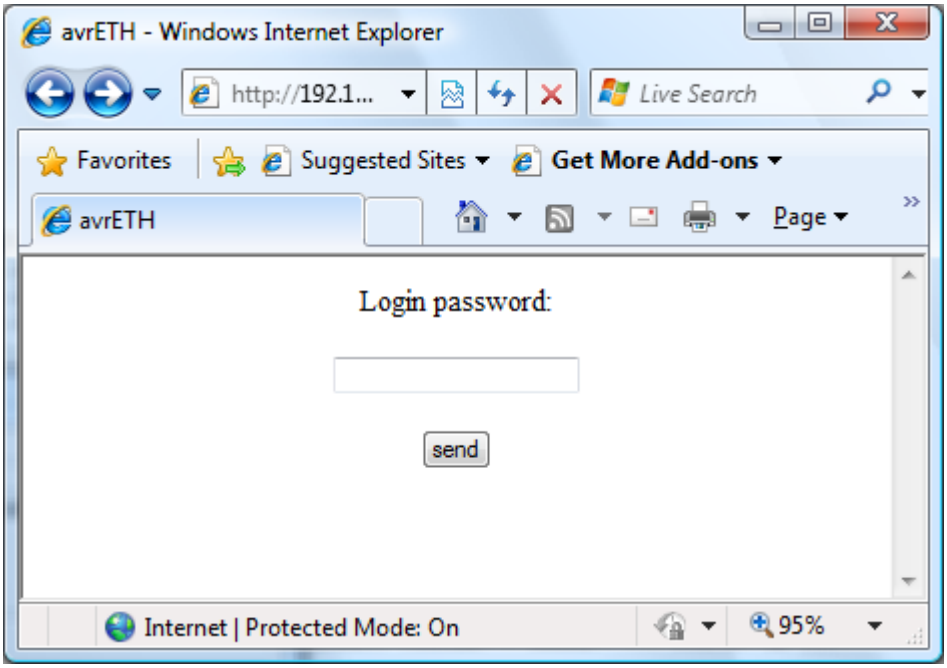
Grafinė aplinka sudaro tokie meniu punktai:

- Įvesties prievadų būsenos;
- Išvesties prievadų būsenos ir valdymas;
- Temperatūros jutiklių parodymų atvaizdavimas;
- RS-232 sąsajos terminalas.

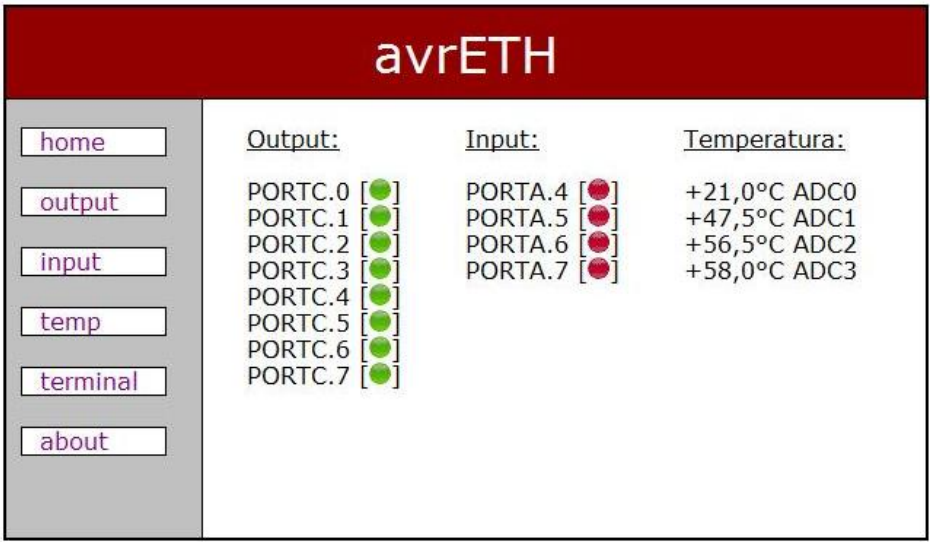
Užklausos mikrovaldikliui formuojamos siunčiant TCP paketus su nuoroda. Mikrovaldiklis gavęs TCP paketą su atitinkamą nuorodą iššifruoją ją ir atlieka funkciją priskirtą šiai nuorodai.

Grafinės naudotojo aplinkos vaizdas yra parodytas (2.13–2.18 pav.). Prisijungti prie sistemos galima tik žinant prisijungimo slaptažodį. Įvedus slaptažodį priėjimas bus leistas vieną minutę, po to vėl reikės įrašyti slaptažodį norint tęsti darbą (2.13 pav.). Pagrindiniame lange (2.14 pav.) matomas bendras vaizdas, Į/Iš prievadų būseną ir temperatūros jutiklių rodmenys. Galima atskirai stebėti įvesties prievadų būsenas (2.15 pav.). Išvesties prievadų

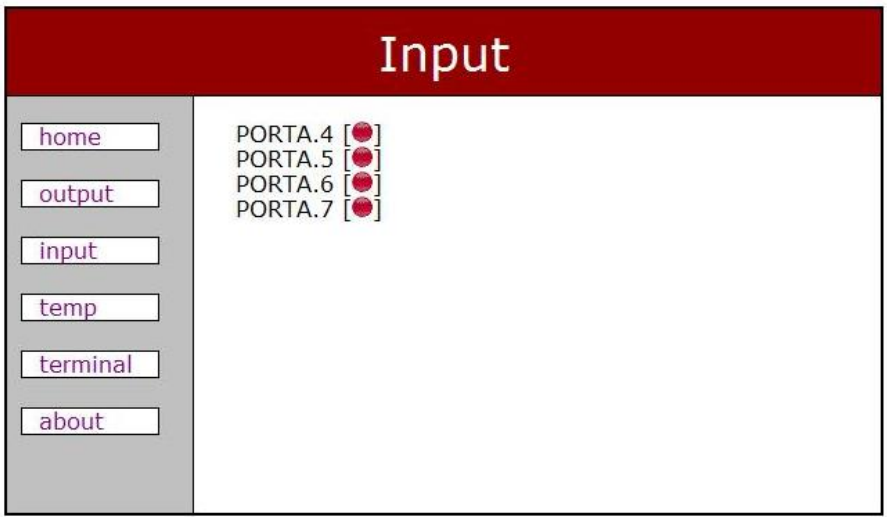
valdymas atliekamas per išvesties valdymo langą (2.16 pav.). Atskiri temperatūros rodmenis išvedami (2.17 pav.) lange. RS-232 sąsajos terminalas (2.18 pav.) leidžia persiusti teksto žinutę per TCP protokolą į RS-232 sąsaja mikrovaldiklyje, tai patogu norint komandomis valdyti įrenginį prijungtą, prie RS-232 sąsajos per *Ethernet* tinklą nuotoliniu būdu. Perėjimas tarp grafinės aplinkos valdymo langų atliekamas mygtukais esančiais kairėje.



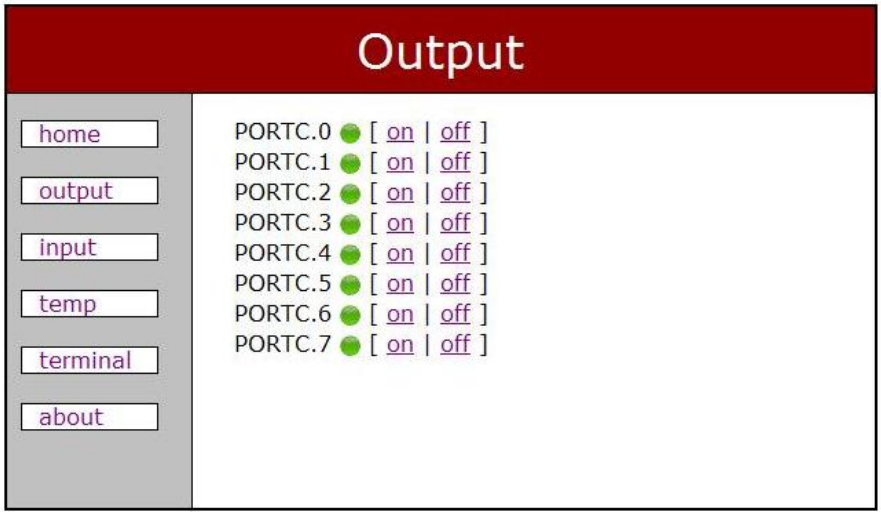
2.13 pav. Prisijungimo prie sistemos langas



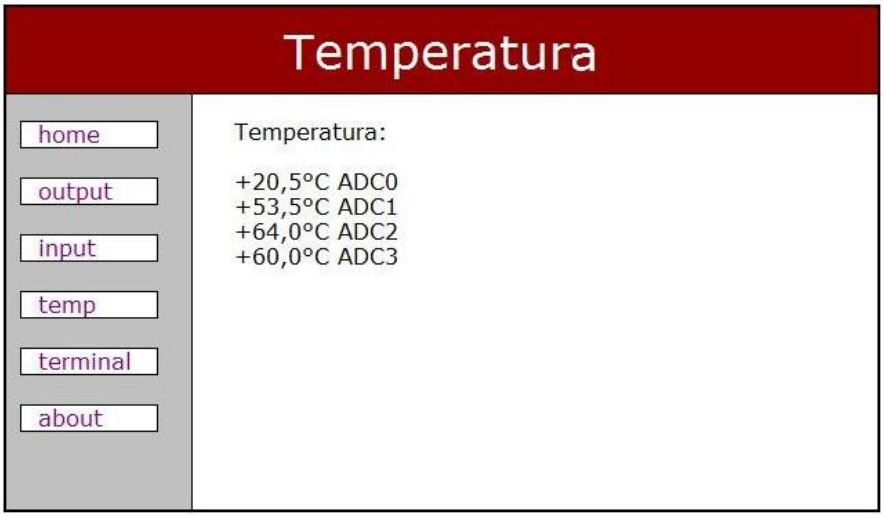
2.14 pav. Pagrindinis valdymo aplinkos meniu



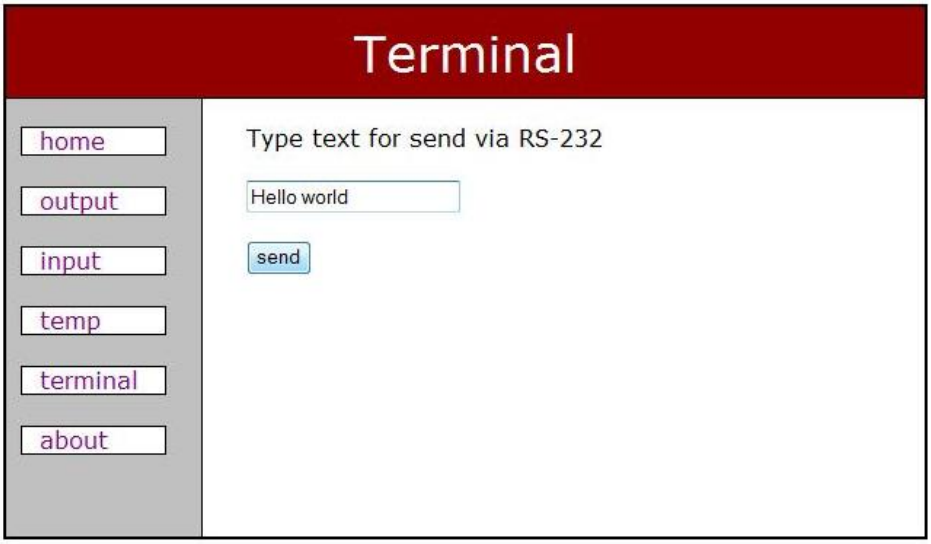
2.15 pav. Įvesties prievadų būsena



2.16 pav. Išvesties prievadų valdymas



2.17 pav. Temperatūros rodmenys



2.18 pav. RS-232 sąsajos terminalo langas

2.4. Programos kodo kompiliavimas ir optimizavimas

Programa mikrovaldikliui bus rašoma *C#* programavimo kalba panaudojus *AVR-GCC* bibliotekas AVR mikrovaldikliams. Programos kodas bus kompiliuojamas *AVR Studio 4* transliatoriuje kartu su *AVR-GCC* kompiliatoriumi. Pasirinkta programavimo kalba *C#* yra aukšto lygio programavimo kalba, to reikalauja sudėtingos funkcijos, kurias reikės atlikti mikrovaldikliui. Nors rašant programą *C#* kalba mikrovaldikliui kodas yra ne optimalus, bus stengiamasi optimizuoti sukompiliuotą kodą, tai leidžia *AVR Studio 4* nustatymai [9, 10]. Yra galimi penki programos kodo optimizavimo lygiai (lentelė 2.2.).

2.2 lentelė. Programos kodo optimizavimo lygiai

Optimizavimo lygis	**Paskirtis**
–O0	Be optimizavimo
–O1	Kodo dydžio optimizavimas. Nesumažina kompiliavimo laiko
–O2	Didesnis optimizavimas, didesnis kodo dydžio ir kompiliavimo laiko santykis
–O3	Dar labiau optimizuojamas kodas, vykdomas –O2 optimizavimas kartu su *-finline-functions* ir *-frename-registers* optimizavimu
–Os	Optimizavimas kodo užimamos atminties. Panaudojamas –O2 optimizavimas, kuris nepadidina kodo dydžio

Buvo padaryti kodo optimizavimo tyrimai ir rezultatai parodyti 2.3 lentelėje. *Atmega32* mikrovaldiklio duomenų atmintis 32 KB, ir operatyvioji atmintis 2 KB. Iš visų išbandytų optimizavimo režimų buvo parinktas –Os optimizavimo lygis.

2.3 lentelė. Programos kodo optimizavimo tyrimas

Optimizavimo lygis	Duomenų atmintis, B	Operatyvioji atmintis, B	Užimta duomenų atmintis, %	Užimta operatyvioji atmintis, %
-O0	37792	1918	115,3	93,7
-O1	21116	1908	64,4	93,2
-O2	20398	1908	62,2	93,2
-O3	33216	1904	101,4	93,0
-Os	19734	1908	60,2	93,2

Lyginant –Os optimizavimo lygį su –O0 optimizavimo lygių, pavyko gauti beveik du kartus mažesnį programos kodo dydį dėl vėlinimo ciklų optimizavimo algoritmo.

Projekte panaudotos *AVR-GCC* bibliotekos:

- `avr/io.h` – pagrindiniai mikrovaldiklio įvesties/išvesties prievadų aprašymai, registrų paskirties aprašymai;
- `stdlib.h` – pagrindinės funkcijos;
- `string.h` – darbas su simbolių sekomis;
- `util/delay.h` – vėlinimo funkcijos;
- `avr/interrupt.h` – pertraukčių funkcijos;
- `avr/pgmspace.h` – programos atminties pasiskirstymo funkcijos;
- `avr/wdt.h` – darbo monitoriaus funkcijos;
- `ip_arp_udp_tcp.h` – TCP/IP protokolo funkcijos;
- `enc28j60.h` – *ENC28J60* valdiklio tvarkyklės;
- `timeout.h` – specialios vėlinimo funkcijos;
- `avr_compat.h` – *C#* kodo suderinamumo biblioteka;
- `net.h` – tinklo nustatymo funkcijos.

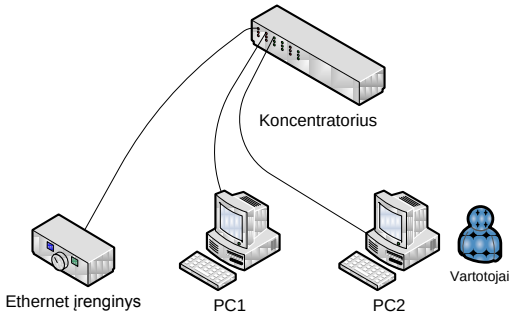
Rašant programą *C#* programavimo kalba labai palengvina jau turimų bibliotekų su funkcijomis panaudojimas. Šiame darbe buvo panaudotos tokios funkcijos kaip vėlinimų, *Ethernet* valdiklio valdymo funkcijos, TCP/IP steko funkcijos, analoginio skaitmeninio keitiklio funkcijos, darbo monitoriaus funkcijos – tai palengvino visos programos rašymą. Pasinaudojant jau sukurtomis funkcijomis nereikėjo rūpintis jų optimizavimu, nes jos buvo jau optimizuotos. Apskritai visame programos kode reikėjo optimizuoti darbinės atminties panaudojimą, mažinant kintamųjų skaičių, mažinti papildomus kreipimus į funkcijas.

3. TYRIMO METODIKOS SUDARYMAS, PRIEMONIŲ PARINKIMAS IR PAGRINDIMAS

Norint ištirti mikrovaldiklių galimybes ir funkcijas panaudojimui kompiuterių tinkluose, bus sudaryta tyrimo metodika ir parinktos atitinkamos priemonės. Tyrimo metodikoje reikia išanalizuoti tokius svarbius tinklo paslaugų parametrus, kaip:

- Reikalavimų mikrovaldikliui tyrimas;
- Saugumo tyrimas;
- Atsparumo DoS atakoms tyrimas;
- Pagrindinių įrenginio tinklo parametrų tyrimas.

Parenkant priemones tyrimui bus atsižvelgiama į tai, kad priemonės būtų lengvai realizuojamos, o tyrimo metodika būtų kuo paprastesnė ir patikimesnė. *Ethernet* įrenginio tyrimo schema pavaizduota (3.1 pav.). Ją sudaro įrenginys su *Ethernet* sąsaja prijungtas prie komutatoriaus. Kompiuteris PC1 kuriame yra paketų generavimo programos ir analizatorius, kompiuteriu PC2 yra valdomas *Ethernet* įrenginys.



3.1 pav. Tyrimo schema

Priėjimas prie įrenginio tikrinamas per kompiuterį PC2. Kompiuteryje PC1 yra įdiegtos paketų generavimo programos *hping3*, atvirų prievadų žvalgos įrankis *nmap*, tinklo paketų analizatorius *Wireshark*. Tam, kad visi kompiuteriai ir įrenginys galėtų matyti tinkle vienas kitą atitinkamai konfigūruojamas komutatorius. Pagal sujungtą schema galimi keli įrenginio tyrimo scenarijai: saugumo tyrimas, atsparumo DoS atakoms tyrimas ir pagrindinių įrenginio *Ethernet* parametrų tyrimas.

3.1. Mikrovaldiklio reikalavimų tyrimas

Įrenginys su galimybe prisijungti prie tinklo turės atitikti minimalius *Ethernet* standarto reikalavimams. Pats žemiausias *Ethernet* fizinio lygmens standartas 10BaseT, kuris atitinka 10 Mbps duomenų perdavimo greitį. Duomenų paketų dydis yra nuo 64 B iki 1518 B. Tam, kad du įrenginiai tinkle galėtų perduoti duomenis vienas kitam, reikia minimalaus TCP/IP protokolų rinkinio, tai yra: ARP, IP, TCP arba UDP. Šis protokolų rinkinys tenkina minimalų reikalavimą įrenginiui prijungti prie tinklo.

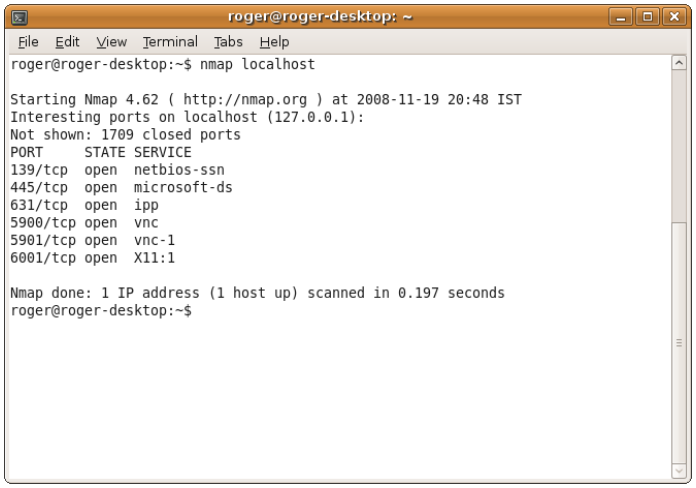
Mikrovaldiklis turi atitikti tokius parametrus:

- Tam, kad *Ethernet* sąsaja pilnai atitiktų 10BaseT standartą, mikrovaldiklis turi perduoti duomenis SPI sąsaja ne mažiau kaip dvigubai didesnių taktinių dažnių, tai yra ne mažiau kaip 20 MHz arba 20 MIPS;
- TCP/IP steko panaudojimui reikia ne mažiau kaip 16 KB programų atminties;
- Duomenų atmintis turi būti ne mažesne kaip 2 KB, nes maksimalus *Ethernet* duomenų paketas gali būti 1518 B dydžio;
- Prisijungimui *Ethernet* valdiklio prie mikrovaldiklio reikia SPI sąsajos.

3.2. Saugumo tyrimas

Ethernet įrenginiai yra jungiami tiesiai prie tinklo ir yra būtinybė patikrinti duomenų perdavimą saugumo požiūriu. Reikia išsiaiškinti šiuos dalykus: 1) kaip yra realizuojamas prisijungimo prie įrenginio procesas; 2) kokia duomenų forma įrenginys apsikeičia ir ar duomenis yra šifruojami; 3) kokius protokolus naudoja įrenginys; 4) kokie transportinio lygmens protokolų prievadai yra atviri; 5) ar įrenginys atsako į ICMP *echo-request* užklausas. Saugumo tyrimas bus atliekamas pagal (3.1 pav.) pateiktą tyrimo schemą.

Tokio tipo tyrimui gerai tinka žvalgos priemonė *nmap*. *Nmap* – (angl. *Network Mapper*) tai yra atviro kodo priemonė skirta tinklo tyrimams ir jo saugumo patikrinimui [12]. *Nmap* panaudojami skirtingų tipų paketai norint nuspręsti, koks mazgas yra pasiekiamas tinkle, kokius servisus (programos pavadinimas ir versija) mazgas naudoja, kokia operacinė sistema naudojama, kokius paketų tipus tarpsegmentinis ekranas praleidžia ir kokius blokuoja, kokie TCP ir UDP prievadai atviri. *Nmap* priemonė gali būti naudojama skirtingose operacinėse sistemose Linux, Windows, MacOS X. Gali būti naudojamas komandinėje eilutėje arba grafinėje aplinkoje. *Nmap* įrankio langas parodytas (3.2 pav.).



3.2 pav. *Nmap* įrankio langas

Viena iš pagrindinių *nmap* funkcijų yra atvirų prievadų paieška. Paieškos rezultatų sąrašo turinyje pateikiamas prievadų sąrašas kartu su naudojamais protokolais, programos pavadinimu bei prievado būsena. Prievado būsenos gali būti: atidarytoji, filtruotoji, uždarytoji, nefiltruotoji.

Atidarytoji – taikomoji programa aktyviai aptarnauja sujungimus TCP arba UDP paketus šiame prievade. Tokių prievadų suradimas yra pagrindinis prievadų žvalgos tikslas. Kadangi atviras prievadas yra potenciali vieta atakai. Atakuojantis gali išnaudoti esamas spragas programose, kurios naudoja šį prievadą.

Uždarytoji – uždarytas prievadas yra pasiekiamas *nmap* priemonei, bet nėra jokios programos, kuri galėtų įvykdyti sujungimą. Toks prievadas gali būti panaudojamas sužinoti ar mazgas yra aktyvus, kai yra blokuojami ICMP *echo-request* paketai. Taip pat naudojami OS versijos atpažinimui. Administratoriai dažnai blokuoja tokius prievadus tarpsegmentiniais ekranais, tada tokį prievadą *nmap* atpažįsta kaip filtruotą.

Filtruotoji – prievadas yra filtruojamas tarpsegmentinio ekrano. Tokio prievado būsenos *nmap* negali nustatyti.

Nefiltruotoji – tai reiškia, kad prievadas yra pasiekiamas, bet *nmap* negali nustatyti jo būsenos. Tokio prievado būsena gali būti aptikta ACK žvalgos būdu.

Nmap žvalgos įrankis bus naudojamas Linux operacinėje sistemoje ir bus panaudota jo komandinės eilutės versija. Žemiau pateikiamas pagrindinių funkcijų sąrašas.

`nmap [žvalgos tipas] [parametrai] {žvalgos taikinys}`

Žvalgos taikinys – galima nurodyti mazgo adresą, mazgo pavadinimą, tinklo adresą.

Pvz.: `192.168.0.1, www.google.lt, 10.0.0-255.1-254.`

`-iL <bylos pavadinimas>` - galima nuskaityti mazgų adresus iš bylos.

Mazgų aptikimas tinkle – naudojamas tam, kad išvestume sąrašą mazgų, esamų tinkle.

`-sL: List Scan` – išvedamas į ekraną žvalgomų mazgų sąrašas;

`-sP: Ping Scan` – išvedamas į ekraną mazgų kurie atsako į ICMP *echo-request* užklausas sąrašas;

`-PS/PA/PU[portlist]` – TCP SYN/ACK ir UDP prievadų aptikimas;

`-PO[protocol list]` – IP protokolo patikrinimas;

`--traceroute`– išvedamas į ekraną tarpinių mazgų sąrašas;

Žvalgos technikos – panaudojami skirtingi žvalgos būdai.

`-sS/sT/sA/sW/sM` - TCP SYN/Connect()/ACK/Window/Maimon žvalga;

`-sU` - UDP skanavimas;

`-sN/sF/sX` - TCP Null, FIN, ir Xmas skanavimas;

`-sO` - IP protokolo žvalga;

Prievadų specifikacija ir žvalgos nuoseklumas – prievadų tipai, galimybe žvalgyti tam tikrus prievadus.

`-p <port ranges>` - nustatomas prievadų intervalas;

Pvz.: `-p22; -p1-65535; -p U:53,111,137,T:21-25,80,139,8080;`

`-F: Fast mode` – žvalgomi populiariausi prievadai;

`-r: Scan ports consecutively` – žvalgoma prievadus nuosekliai;

Programų versijų atpažinimas – galimybė atpažinti naudojamas programas.

`-sV` – tikrinant atidarytus prievadus bandoma nustatyti programos pavadinimą;

OS versijos atpažinimas – galimybė atpažinti naudojamos OS versiją.

`-O` – tikrinama OS versija;

Tarpsegmentinio ekrano ir IDS išvengimas bei spoofing – priemonės naudojamos spoofingui, bei tarpsegmentinio ekrano apėjimui.

`-f; --mtu <val>` – įjungti paketų fragmentavimą, nustatyti MTU dydį;

`-S <IP_Address>` – pakeičia siuntėjo IP adresas;

`-g/--source-port <portnum>` – pakeičia siuntėjo prievado numerį;

`--data-length <num>` – atsitiktinis paketų dydis;

`--ip-options <options>` – galimybė pakeisti IP požymio žymę;

`--ttl <val>` – galimybė nustatyti TTL reikšmę;

`--spoof-mac <mac address/prefix/vendor name>` – MAC adreso pakeitimas;

`--badsum` – galimybė siųsti TCP/UDP paketus su blogomis kontrolinėmis sumomis;

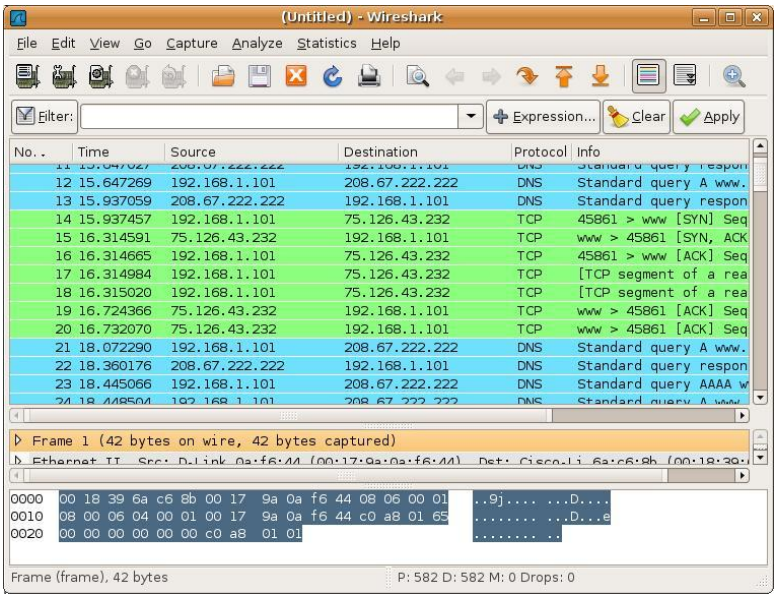
Duomenų srauto sekimui ir paketų analizei bus naudojamas *Wireshark* paketų ir protokolų analizatorius.

Wireshark – yra tinklo paketų analizatorius [14]. Naudojant šią programą galima įrašyti paketus keliaujančius tinkle ir parodyti jų turinį, nustatyti protokolo tipą. Tai yra vienas iš populiariausių atviro kodo paketų analizavimo įrankių. *Wireshark* programos langas parodytas (3.3 pav.). Paketų analizatorius yra naudojamas šiems tikslams:

- Analizuoti problemas tinkle;
- Analizuoti saugumo problemas;
- Tyrinėti, kurti ir tobulinti protokolus;
- Kaip mokomoji priemonė protokolams studijuoti;

Wireshark tinklo paketų analizatorius pasižymi tokiom savybėm;

- Galimybė dirbti *Linux* ir *Windows* aplinkoje;
- Galimybė matyti ateinančius paketus realiame laike;
- Galimybė išsaugoti paketus;
- Paketų filtravimas pagal kriterijus;
- Įvairių statistikų atvaizdavimas.



3.3 pav. *Wireshark* programos langas

Pasinaudodami *nmap* ir *Wireshark* programomis galime ištirti įrenginių saugumą. Tyrimas bus atliekamas pagal (3.1 pav.) tyrimo schemą. Saugumo tyrimo nuoseklumas aprašytas žemiau.

1. Iš kompiuterio PC2 bandoma prisijungti prie valdomo įrenginio. PC1 kompiuteryje paleidžiamas *Wireshark* paketų analizatorius. Įrašomi visi perduoti prisijungimo sesijos paketai ir analizuojami. Bandoma pamatyti prisijungimo vardą ir slaptažodį.
2. Iš kompiuterio PC2 bandoma valdyti įrenginį. PC1 kompiuteryje paleidžiamas *Wireshark* paketų analizatorius. Įrašomi visi perduoti įrenginio valdymo sesijos paketai ir analizuojami. Bandoma nustatyti duomenų pateikimo formatą valdomajam įrenginiui.
3. Kompiuteryje PC1 paleidžiamas *nmap* žvalgos įrankis ir atliekami valdomojo įrenginio testavimai. Nustatomi atviri TCP ir UDP prievadai. Nustatoma šių prievadų paskirtis. Bandoma nustatyti naudojamus protokolus, OS versija, naudojamas programas.
4. Pagal gautus rezultatus iš 3 punkto bandoma rasti saugumo spragas. Bandoma prisijungti ir valdyti įrenginį.
5. Padaromos saugumo tyrimo išvados, nurodomos neigiamybės bei pažeidimai.

3.3. Atsparumo DoS atakoms tyrimas

Išanalizavus įrenginio panaudojimo galimybes ir naudojamus protokolus, galima pradėti atsparumo DoS (angl. *Denial of Service – DoS*) atakoms tyrimą. Tokio tyrimo tikslas nustatyti *Ethernet* įrenginio procesoriaus pajėgumą apdoroti didelį kiekį duomenų paketų [15]. Jungiant įrenginį prie globalaus tinklo į tai reikia atkreipti ypatingą dėmesį, kadangi internete atsisakymo aptarnauti atakos tampa vis labiau įprastos ir dažnos. Be to, didelio apkrovimo tinkluose tinklo mazgams tenka apdoroti didelį kiekį visiems skirtų (angl. *Broadcast*) paketų. DoS atakos pagrindinis tikslas paveikti kompiuterinę sistemą arba tinklą taip, kad paslaugos taptų nepasiekiamos naudotojams. DoS atakos metu gali būti perkrauti kompiuterio, įrenginio ištekliai (perpildytas tinklas, apkrauta atmintis, CPU ir pan.). Tai atsitinka dėl to, kad sistema bando apdoroti nereikalingą arba klaidingą informaciją, išnaudoja daug savo išteklių ir nepajėgia aptarnauti autorizuotų naudotojų, siunčiančių užklausas.

DoS atakos dažniausiai realizuojamos *flood* paketų atakomis. *Flood* ataka realizuojama siunčiant specialiai modifikuotas užklausas, tai priveda prie procesoriaus apkrovimo, atminties išnaudojimo arba ryšio kanalo perpildymo.

Flood atakos gali būti tokios:

- SYN ataka – tokia ataka realizuojama siunčiant didelį kiekį SYN paketų TCP protokolų [16]. Tokia ataka paremta TCP protokolo sesijos užmezgimo ypatybėmis. Pagal TCP protokolo taisykles du mazgai (klientas ir serveris) prieš pradedant

duomenų apsikeitimo procesą turi inicijuoti tarpusavio ryšį, tai yra jie turi persiųsti preliminarius paketus vienas kitam, kuriais nurodytų ryšio parametrus ir užtikrintų duomenų perdavimą. Pirmiausia klientas serveriui siunčia paketą su nurodytų SYN požymiu, serveris atsako SYN|ACK paketu ir pabaigoje klientas patvirtina sesijos užmezgimą ACK paketų. Toks ryšio užmezgimo procesas vadinasi trijų dalių rankos paspaudimu (angl. *three-way handshake*). Dėl tokios atakos po kažkiek laiko įrenginyje gali neužtekti ištieklių apdoroti tokias užklausas dėl didelio sesijų skaičiaus. Įrenginys tampa nepasiekiamas.

- UDP ataka – tokios atakos tikslas yra ryšio kanalo perpildymas [17]. Daugeliu atvejų UDP duomenų srautas yra aukštesnio prioriteto negu TCP srautas. Dėl to serveris arba įrenginys dirbantis su TCP protokolų gali nustoti veikti. Tokia ataka orientuota į tinklo perpildymą.
- ICMP atakos – tokia pat ataka kaip ir UDP bet su ICMP paketais. Taip pat yra ir kitoks ICMP atakos būdas. Kai yra formuojamas ICMP PING paketas kur gavėjo adresas yra viešas (angl. *Broadcast*) adresas, o siuntėjo adresas pakeičiamas į norimo atakuoti serverio adresą. Tokių atveju visi įrenginiai kurie gavo ICMP užklausą atsako į ją, dėl to gaunamas labai didelis IP paketų šuolis link atakuojamo serverio.

DoS atakų sudarymui bus naudojama *hping3* programa. *Hping3* – tai yra TCP/IP paketų generavimo ir analizės įrankis [11]. Įrankis palaiko tokius protokolus kaip TCP, UDP, ICMP ir RAW-IP. Žemiau pateiktas pagrindinių funkcijų sąrašas.

Hping3 įrankio panaudojimo galimybės:

- Tarpsegmentinio ekrano testavimas;
- TCP ir UDP prievadų žvalga;
- Tinklo pralaidumo tikrinimas, panaudojus skirtingus protokolus, skirtingus paketų dydžius, paketų TOS žymės panaudojimas, fragmentacija;
- Maršruto MTU aptikimas;
- Duomenų perdavimas per stipriai apsaugotus tarpsegmentinius ekranus;
- *Traceroute* komandos panaudojimas su skirtingais protokolais;
- Nutolusio mazgo OS sistemos atpažinimas;
- TCP/IP steko auditas;
- Galimybė panaudoti *hping3* kartu su TCL skriptu.

Hping3 įrankis bus naudojamas Linux operacinėje sistemoje ir bus panaudota jo komandinės eilutės versiją.

`Hping3 [taikinys] [parametrai]`

Protokolo nustatymai:

Protokolas pagal nutylėjimą yra nustatytas TCP.

`--icmp` – ICMP režimas, yra formuojamos ICMP echo-request užklausos, yra galimybė nustatyti kitą ICMP paketo tipą;

`--udp` – UDP režimas, pagal nutylėjimą yra formuojami UDP paketai su nulinių prievadu;

`--scan` – žvalgos režimas;

`--listen` – galimybė įjungti paketo laukimo režimą pagal požymį;

Papildomi nustatymai:

`-i` – paketų siuntimo periodo nustatymas;

`-a` – galima keisti siuntėjo IP adresą;

`--rand-source` – atsitiktinis siuntėjo IP adreso siuntimas;

`--rand-dest` – atsitiktinis gavėjo IP adreso siuntimas;

`-t` – TTL reikšmės nustatymas;

`-f` – įjungiamas paketų fragmentavimas;

`-y` – išjungiamas paketų fragmentavimas;

`-m` – nustatoma MTU reikšmė;

`-o` – nustatoma TOS reikšmė;

`-T` – traceroute komandos panaudojimas;

ICMP/TCP/UDP nustatymai:

`-C` – ICMP tipo nustatymas, pagal nutylėjimą yra nustatytas ICMP echo-request tipas;

`-K` – ICMP kodo reikšme;

`-s` – siuntėjo prievadas;

`-p` – gavėjo prievadas;

`-b` – blogos kontrolinės sumos nustatymas;

`-S` – SYN žymės nustatymas;

`-R` – RST žymės nustatymas;

`-P` – PUSH žymės nustatymas;

`-A` – ACK žymės nustatymas;

`-U` – URG žymės nustatymas;

`-X` – Visų TCP žymių nustatymas;

`-Y` – Visos TCP žymės su reikšme nulis;

`-d` – duomenų paketo dydžio nustatymas;

Pasinaudodami *hping3* programa galime ištirti įrenginio atsparumą DoS atakoms. Tyrimas bus atliekamas pagal (3.1 pav.) tyrimo schemą. Atsparumo DoS atakoms tyrimo nuoseklumas aprašytas žemiau.

1. Pagal saugumo tyrimo gautus rezultatus, turint informaciją apie atidarytus TCP ir UDP prievadus yra formuojama atitinkama DoS ataka, parenkamas jos tipas bei realizavimo galimybė.
2. Mazgo pasiekimas yra tikrinamas keliais būdais, bandant prisijungti prie jo ir tikrinant jo galimybę atsakyti į ICMP *echo-request* užklausas, jeigu bent vienas iš mazgo pasiekimo būdų neveikia galima laikyti, kad DoS ataka yra sėkminga.
3. Atidarytiems TCP prievadams formuojama SYN *flood* ataka;
4. Atidarytiems UDP prievadams formuojama UDP *flood* ataka;
5. Jeigu įrenginys atsako į ICMP *echo-request* užklausas, tada yra formuojama ICMP ataka;
6. Analizuojami gauti rezultatai, formuojama DoS atakos realizavimo ataskaita, padaromos išvados.

3.4. Įrenginio pagrindinių tinklo parametrų tyrimas

Ethernet technologija yra paremta paketų komutavimo principu. Duomenys tinkle yra perduodami paketais. Paketas tinkle gali keliauti skirtingais maršrutais, nes turi savyje tokią informaciją kaip gavėjo adresas ir siuntėjo adresas. Taip pat yra svarbu, kad duomenys pasiektų gavėją. Tam yra sukurti duomenų pasiekimo mechanizmai, tokie kaip patvirtinimo siuntimas apie gautus duomenis, paketai yra numeruojami, tikrinamas jų vientisumas. Tokiame paketiniame duomenų perdavimo tinkle yra svarbūs šie parametrai: paketo dydis, perdavimo sparta, paketų vėlinimas, perdavimo režimas, paketų praradimas. Tinklo įrenginiams yra svarbūs tokie parametrai kaip: duomenų apdorojimo sparta, maksimalus paketo dydis, paketų vėlinimas, perdavimo režimas, buferio dydis.

Visų šitų parametrų nustatymui yra daug skirtingų būdų. Taip pat yra apibendrinta tinklo parametrų nustatymo metodika kuri yra aprašyta RFC2544 standarte [8]. RFC2544 tai yra

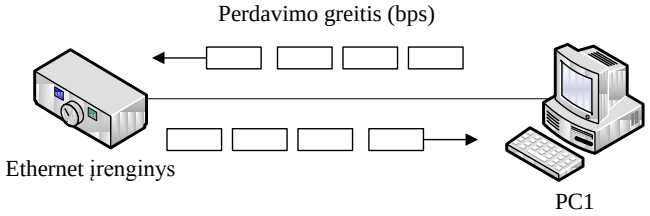
Ethernet tinklo sertifikavimo standartas. Jame yra nurodyta *Ethernet* tinklo parametrų nustatymo metodika.

Trumpai apžvelgsime tyrimo metodikas bei standartus.

1. Testavimas turi būti atliekamas pagal tokią matavimo schemą, kad tinklo įrenginys butų betarpiškai prijungtas prie matavimo įrenginio. Mūsų atvejų matavimo įrenginio funkcijas atliks kompiuteris PC1. Matavimo schema parodyta (3.4 pav.).

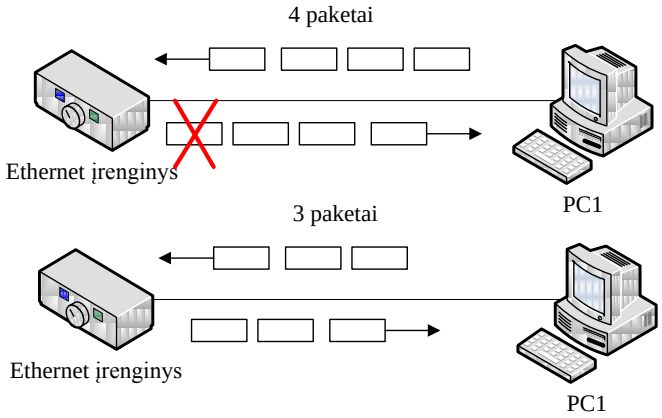
3.4 pav. Matavimo schema

2. *Ethernet* tinklo matavimams panaudojami standartizuoti paketų dydžiai: 64, 128, 256, 512, 1024, 1280 ir 1518. Tokie paketų dydžiai apima mažiausią paketo dydį 64 B ir didžiausią paketo dydį 1518 B.
3. Maksimali duomenų apdorojimo sparta yra matuojama skirtingiems paketų dydžiams. Paketai yra siunčiami postoviu greičiu. Matavimo principas parodytas (3.5 pav.).



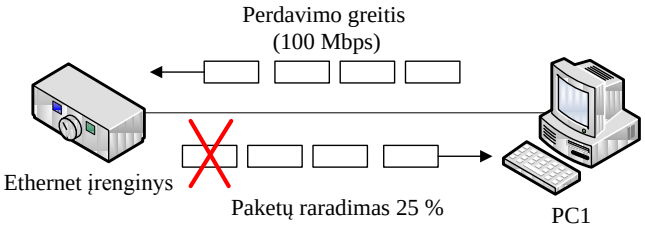
3.5 pav. Maksimali duomenų perdavimo sparta

4. Maksimalus apdorojamų paketų skaičiaus matavimas (angl. *Packet Burst*). Nustatomas koks maksimalus yra duomenų paketų kiekis kuri įrenginys gali apdoroti. Tai priklauso nuo minimalaus atstumo nuo paketų. Toks matavimo principas parodytas (3.6 pav.).



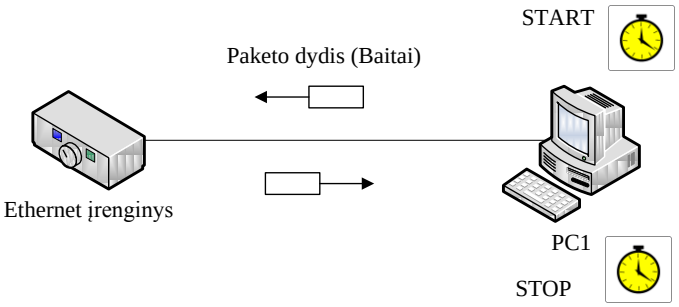
3.6 pav. Duomenų paketų šuoliai

5. Paketų praradimo matavimas atliekamas prie tam tikros duomenų perdavimo spartos. Ir nustatomas procentinis duomenų paketų praradimas. Matavimas atliekamas su skirtingais paketų dydžiais. Matavimo principas parodytas (3.7 pav.).



3.7 pav. Paketų praradimas

6. Paketų vėlinimas yra matuojamas prie skirtingų paketų dydžių. Nustatoma kaip priklauso vėlinimas nuo pakėto dydžio. Paketų vėlinimo matavimo principas parodytas (3.8 pav.).



3.8 pav. Paketų vėlinimas

7. Nustatomas įrenginio duomenų paketų apdorojimo buferio dydis. Jį galima nustatyti tik iš *Ethernet* valdiklio dokumentacijos. Buferis naudojamas duomenų paketų saugojimui prieš jų apdorojimą.
8. Nustatomas įrenginio duomenų perdavimo režimas. Perdavimo režimai gali būti *Full Duplex* arba *Half Duplex.*
9. Nustatoma įrenginio naudojama *Ethernet* technologija, gali būti 10BaseT, 100BaseT, 1000BaseT. Parametras nustatomas įrenginio prisijungimo prie kompiuterio metu.

Tyrimui atlikti bus panaudotas paprasčiausias ICMP paketų generatorius *ping*. *Ping* įrankis bus naudojamas Linux operacinėje sistemoje ir bus panaudota jo komandinės eilutės versiją [5, 6]. Žemiau pateiktas pagrindinių funkcijų sąrašas.

```
ping [taikinys] [parametrai]
```

Parametrai:

`-b` – leidžiamas 255.255.255.255 (*angl. broadcast*) adresų paketų siuntimas;

`-c` – paleidžiamas n paketų skaičius;

`-f` – paketų siuntimas maksimalia sparta;

`-i` – intervalo tarp paketų nustatymas ms;

`-Q` – nustatoma TOS reikšmė;

`-s` – nustatomas paketo dydis;

`-t` – nustatoma TTL reikšmė;

`-U` – grąžinama tikroji paketo vėlinimo reikšmė;

`-W` – nustatomas ICMP *echo-replay* laukimo laikas;

Šiame skyriuje buvo sudaryta įrenginių tyrimo metodika. Numatytos tyrimo sritys tokios kaip saugumo tyrimas, atsparumo atakoms tyrimas, pagrindinių įrenginio parametrų tyrimas. Toliau darbe išsamiai ištirsime aštuonių skilčių mikrovaldiklio parametrus ir jo panaudojimo *Ethernet* tinkle galimybes.

4. ĮRENGINIŲ TYRIMAS

Šiame skyriuje pateikti *Ethernet* tinklo įrenginių tyrimo rezultatai. Tyrimas apims paketų apdorojimo spartos nustatymą skirtingiems TCP/IP lygmenų protokolams, o būtent ICMP tinklo lygmens protokolas, UDP transporto lygmens protokolas, bei TCP ir HTTP protokolų kompleksinis tyrimas. ICMP protokolas parinktas testavimui kaip tinklo lygmens protokolas, jo apdorojimas vyksta tinklo lygmenyje, dažniausiai naudojamas įrenginių tinkle pasiekiamumo nustatymui, naudojamas nustatyti RTT (angl. *Round-Trip Time*) vėlinimo reikšme. UDP protokolas parinktas kaip paprasčiausias transporto lygmens protokolas, jo naudojimas nereikalauja sesijos užmezgimo, plačiai naudojamas realaus laiko uždavinių sprendimui. TCP yra transporto lygmens protokolas, jo veikimas bus tikrinamas kartu su HTTP taikomojo lygmens protokolu. Šie abu protokolai reikalauja sesijos užmezgimo. Tyrimo metu tikimasi gauti skirtingų protokolų duomenų maksimalios apdorojimo spartos, vėlinimo, paketų praradimo tikimybės nuo paketų siuntimo greičio rezultatus. Tyrimai bus atliekami skirtingiems paketų dydžiams pagal RFC 2544 rekomendaciją (64, 128, 256, 512, 1024, 1280, 1518 B). Visi tyrimai bus atliekami *Linux* operacinės sistemos aplinkoje. Naudojamos priemonės: paketų generatoriai *Hping3, Ping*; paketų analizavimo priemonė *Wireshark*; paketų atkartojimo priemonė *TCPreplay*.

Tyrimui buvo parinkti įrenginiai su skirtingais mikrovaldikliais (*Atmega32, ATmega128, PIC16F877A, dsPIC30F6014A*). Visiems matavimams kaip pagrindas buvo parinktas bendras *Mikrochip ENC28J60 Ethernet* valdiklis. Naudojamų mikrovaldiklių pagrindinės techninės charakteristikos pateiktos 4.1 lentelėje.

4.1 lentelė Mikrovaldiklių pagrindinės charakteristikos

Mikrovaldiklis	**Skiltiškumas, Bitai**	**SRAM, Baitai**	**Taktinis dažnis, MHz**	**Programų atmintis, Kilobaitai**	**EEPROM atmintis, Baitai**
ATmega32	8	2048	16	32	1
ATmega128	8	4096	10	128	4
PIC16F877A	8	368	10	14,3	256
dsPIC30F6014A	16	8192	10	144	4

Skirtingos mikrovaldiklių konfigūracijos bus naudojamos skirtingiems tyrimams, kurie bus aptarti žemiau.

Tyrimams panaudotas bendras *Ethernet* valdiklis, tam kad eliminuoti galimą įtaką tyrimo rezultatams. Skirtingų gamintojų *Ethernet* valdikliai turi skirtingą architektūrą ir

skirtingas funkcijas. Tyrimo tikslas – ištirti mikrovaldiklius. Parinktas *Mikrochip ENC28J60* valdiklis. *Ethernet* valdiklio techninės charakteristikos pateiktos 4.2. lentelėje.

4.2. lentelė. Ethernet valdiklio charakteristikos

	ENC28J60
Ethernet technologija	10BaseT
Perdavimo režimas	Full/Half Duplex
Buferio dydis	8 Kilobaitai
Taktinis dažnis	25 MHz
Aparatinis apdorojimas	MAC/PHY

Tyrimo darbų palengvinimui buvo suvienodinti įrenginių parametrai. Įrenginių IP adresas parinktas 192.168.20.60 ir fizinis MAC adresas parinktas 00:14:A5:76:19:3F. Šių parametrų suvienodinimas leis naudoti tuos pačius sugeneruotus paketus visiems tyrimams, apie tai bus rašoma žemiau.

4.1. Tyrimo protokolai

ICMP protokolo tyrimui panaudosime protokolo funkciją leidžiančia nustatyti ar mazgas yra pasiekiamas tinkle. Tai yra ICMP *echo-request* užklausos siunčiamos mazgui į kurias mazgas atsako ICMP *echo-replay* paketais. ICMP užklausos siuntėjas taip pat gauną naudingą informaciją apie paketo kelionės laiką iki mazgo ir atgal. ICMP protokolas yra tinklo lygmens protokolas. ICMP *echo-request* žinutė yra formuojama pasinaudojus *ping* paketų generavimo priemonė. *Ping* priemonės nustatymai buvo aptarti ankstesniame skyriuje. ICMP *echo-request* paketo generavimo nustatymai parodyti žemiau [5, 6].

```
ping 192.168.20.60 –s 64 –c 1
```

Formuodami po vieną paketą skirtingo dydžio, įrašome į atskiras bylas panaudojus *Wireshark* paketų analizatorių, šie paketai bus reikalingi tolimesniam tyrimui ir jų daugkartiniam atkartojimui.

Tyrimo metu yra generuojamas ICMP *echo-request* paketų srautas atkartojant iš išsaugotų bylų panaudojus *TCPreplay* paketų atkartojimo priemonę. *TCPreplay* priemonė buvo aptarta ankstesniame skyriuje. *TCPreplay* komandinės eilutės pavyzdys atkartojant 64 B paketą 1 Mbps greičiu per *eth0* kompiuterio tinklo adapterį:

```
TCPreplay –i eth0 –K –q –l 0 –M 1.0 ping_64B
```

Siunčiamų paketų srautas ir gaunamų atsakymo paketų srautas yra fiksuojamas *System Monitor* programos lange. Taip yra gaunamas įrenginio ICMP protokolo apdorojimo greitis.

Kitas pasirinktas tyrimo protokolas yra UDP transporto lygmens protokolas. Tyrimui bus naudojamas įrenginyje padarytas funkcionalumas į atėjusius UDP paketus atsakyti tuo pačiu paketu. Įrenginys gavęs UDP paketą pakeičia siuntėjo ir gavėjo IP adresus vietomis apskaičiuoja kontrolinės sumas ir persiunčia paketą atgal nekeisdamas jo turinio. Tyrimo metu taip pat yra fiksuojamas paketo kelionės laikas iki mazgo ir atgal. UDP paketai bus formuojami panaudojus *Hping3* paketų generavimo priemonę. *Hping3* priemonės nustatymai buvo aptarti ankstesniame skyriuje. Generuojant UDP paketą parenkami tie patys gavėjo ir siuntėjo prievadai. Žemiau parodytas *Hping3* komandinės eilutės pavyzdys generuojant UDP paketą [5, 6].

```
hping3 --udp -c 1 -I eth0 -s 1200 -p 1200 -d 64
192.168.20.60
```

Formuodami po vieną skirtingo dydžio paketą, įrašome į atskiras bylas panaudojus *Wireshark* paketų analizatorių, šie paketai bus reikalingi tolimesniam tyrimui ir jų daugkartiniam atkartojimui.

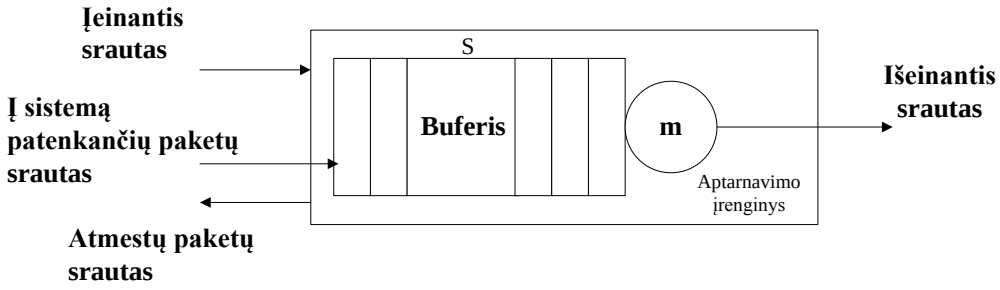
Tyrimo metu yra generuojamas UDP paketų srautas atkartojant iš išsaugotų bylų panaudojus *TCPreplay* paketų atkartojimo priemonę. *TCPreplay* komandinės eilutės pavyzdys atkartojant 64 B paketą 1 Mbps greičiu per *eth0* kompiuterio tinklo adapterį.

```
TCPreplay –i eth0 –K –q –l 0 –M 1.0 udp_64B
```

Siunčiamų paketų srautas ir gaunamų atsakymo paketų srautas yra fiksuojamas *System Monitor* programos lange. Taip yra gaunamas įrenginio UDP protokolo apdorojimo greitis.

4.2. Masinio aptarnavimo teorijos modelis

Suprojektuotas įrenginys buvo sumodeliuotas kaip vienkanalė masinio aptarnavimo sistema (MAS) su apribota eile [2, 3]. Vienkanalės masinio aptarnavimo sistemos modelis pateiktas (4.1 pav.).



4.1 pav. Vienkanalė MAS su apribota eile

Pagrindinės sistemos darbą apibrėžiančios charakteristikos

- sistemos panaudojimo faktorius:

$$\rho = \frac{\lambda}{\mu}, \qquad (1)$$

čia: μ - paketo aptarnavimo sparta; λ - ateinančių paketų intensyvumas;

- paketų atmetimo tikimybė:

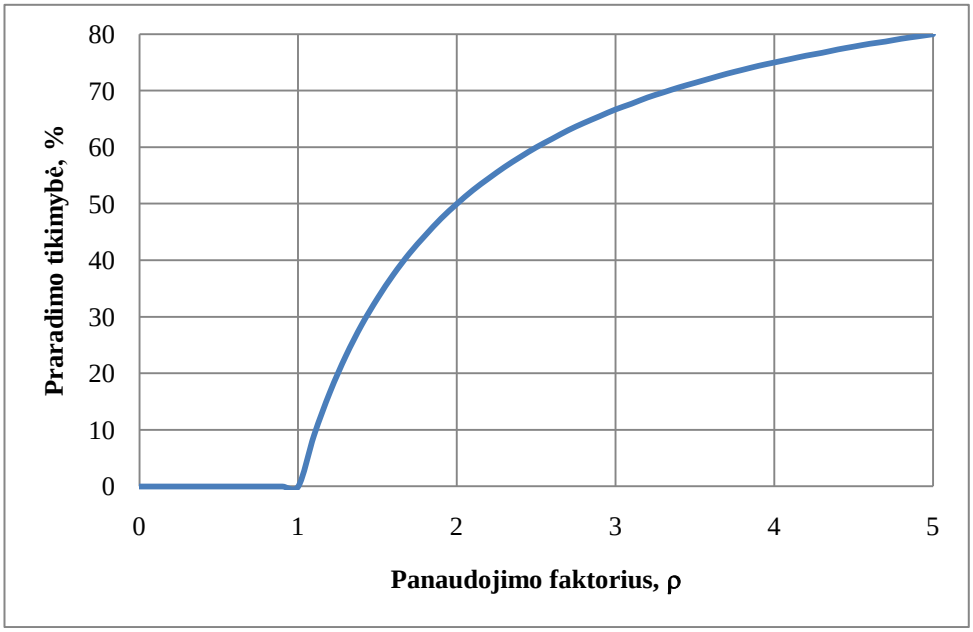
$$P_S = \frac{(1-\rho)\rho^S}{1-\rho^{S+1}}, \qquad (2)$$

čia: S – buferio dydis;

- vidutinis eilėje laukiančių paketų skaičius:

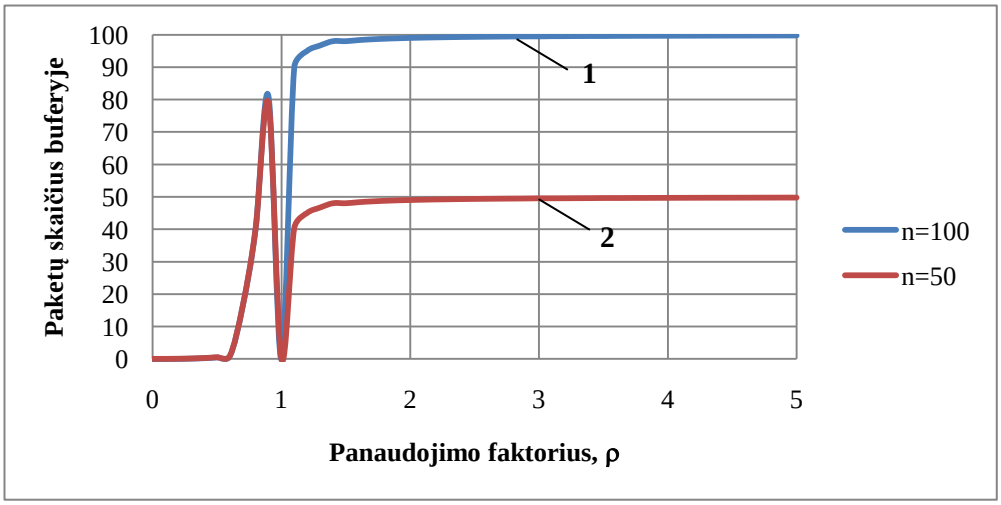
$$N_q = \frac{\rho^2}{1-\rho} - \frac{\rho(S+\rho)}{1-\rho} \cdot P_S, \qquad (3)$$

Paketų atmetimo tikimybė nepriklauso nuo buferio dydžio ir nuo paketų dydžio, tačiau priklauso nuo panaudojimo faktoriaus. Teoriškai apskaičiuota paketų praradimo priklausomybė, kai buferio dydis n = 50 ir n = 100 (4.2 pav.). Iš grafiko matome, kad didinant sistemos panaudojimo faktorių nuo 0 iki 1 sistema apdoroja visus ateinančius paketus. Kai sistemos panaudojimo faktorius viršiją >1 sistema nebespėja apdoroti paketų ir dalį jų atmeta. Praradimo tikimybė didinant panaudojimo faktorių didėja eksponentinių dėsniu.



4.2 pav. Paketų praradimo tikimybė, kai buferio dydis n = 50 ir n = 100

Buferio užpildymo priklausomybė nuo sistemos panaudojimo faktoriaus parodyta (4.3 pav.).



4.3 pav. Vidutinis paketų skaičius buferyje,
1 — kai buferio dydis n = 100, 2 — kai buferio dydis n = 50

Vidutinis paketų skaičius eilėje priklauso nuo sistemos panaudojimo faktoriaus, kai panaudojimo faktorius > 1 buferis staigiai persipildo, o persipildymo greitis mažai priklauso nuo buferio dydžio.

4.3. Tyrimo rezultatai

Norint palyginti aštuonių skilčių mikrovaldiklių spartą buvo sudaryti šeši tyrimai. Jie apima tyrimus panaudojant skirtingas programavimo kalbas, skirtingus programų algoritmus ir skirtingus mikrovaldiklius. Trumpas tyrimų aprašymas ir tikslas pateikti žemiau.

1. Suprojektuoto įrenginio *ATmega32@16Mhz* paketų apdorojimo spartos tyrimas. Apdorojimo spartos didinimas nekeičiant programos algoritmo, tačiau keičiant kompiliatoriaus nustatymus.
2. Panaudojus tą pačią aparatinę bazę, tą patį mikrovaldiklį ištirti paketų apdorojimo spartos priklausomybę nuo panaudoto programos algoritmo. Palyginti dvi programas parašytas panaudojus skirtingus kompiliatorius *AVR-GCC* ir *mikroC*.
3. Dviejų skirtingų mikrovaldiklių tyrimas (*ATmega128@10MHz* ir *PIC16F877A@10MHz*) panaudojus vienodą programos algoritmą, programavimo kalbą ir kompiliatorių.
4. *PIC16F877A@10MHz* mikrovaldiklio paketų apdorojimo spartos tyrimas panaudojus, vienodą programos algoritmą parašytą skirtingomis programavimo kalbomis: *mikroC, mikroPascal* ir *mikroBasic*.
5. Aštuonių skilčių mikrovaldiklių paketų apdorojimo spartos rezultatų palyginimas su 16 skilčių mikrovaldikliu *dsPIC30F6014A@10MHz*.
6. TCP ir HTTP protokolų kompleksinis tyrimas panaudojus *ATmega32@16Mhz* mikrovaldiklį.

Tiriamųjų stendų konfigūracijos ir panaudotos programavimo kalbos parodytos 4.3. lentelėje.

4.3 lentelė. Tiriamieji stendai

Nr.	Mikrovaldiklis	Programavimo kalba	Tyrimas					
			1	2	3	4	5	6
1	ATmega32@16Mhz	C# (–Os)	■					
2	ATmega32@16MHz	C# (–O3)	■					
3	ATmega128@10MHz	C# (–Os)		■				
4	ATmega128@10MHz	mikroC		■	■			
5	PIC16F877A@10MHz	mikroC			■	■		
6	PIC16F877A@10MHz	mikroPascal				■		
7	PIC16F877A@10MHz	mikroBasic				■		
8	dsPIC30F6014A@10MHz	mikroPascal					■	
9	ATmega32@16Mhz	C# (-Os)						■

Pirmas Tyrimas

Suprojektuoto įrenginio ATmega32@16Mhz paketų apdorojimo spartos tyrimas. Apdorojimo spartos didinimas nekeičiant programos algoritmo, tačiau keičiant kompiliatoriaus nustatymus. Lyginant mikrovaldiklio duomenų apdorojimo spartą naudojant -O0, –O1, –O2 ir –O3 programos kodo optimizavimo lygius, jokio programos veikimo spartos pokyčių nepastebėta. Didžiausias skirtumas mikrovaldiklio duomenų apdorojimo spartoje, yra naudojant –O3 ir –Os programos kodo optimizavimo lygius. Lygis –Os geriausiai suspaudžia programos kodą dėl vėlinimo ciklų kodo optimizavimo. Lygis –O3 mažiau suspaudžia programos kodą. Pagrindinis mikrovaldiklio paketų apdorojimo spartos tyrimas buvo atliktas panaudojus –Os kompiliatoriaus kompiliavimo lygį. Gauti rezultatai ICMP ir UDP protokolams parinkus –Os kompiliavimo lygį parodyti 4.4 lentelėje.

4.4 lentelė. *ATmega32@16Mhz* paketų apdorojimo rezultatai, kai naudojamas –Os kompiliavimo lygis

Paketų dydis, B	**ICMP protokolas apdorojimo sparta, Mbps**	**ICMP protokolas paketų vėlinimas, ms**	**UDP protokolas apdorojimo sparta, Mbps**	**UDP protokolas paketų vėlinimas, ms**
64	1,4	0,836	1,3	1,1
128	1,6	1,242	1,5	1,4
256	1,8	1,943	1,6	2,2
512	2,0	3,344	1,7	3,8
1024	2,0	6,165	1,7	7
1280	2,1	7,56	1,8	8,6
1518	2,2	8,494	1,8	9,6

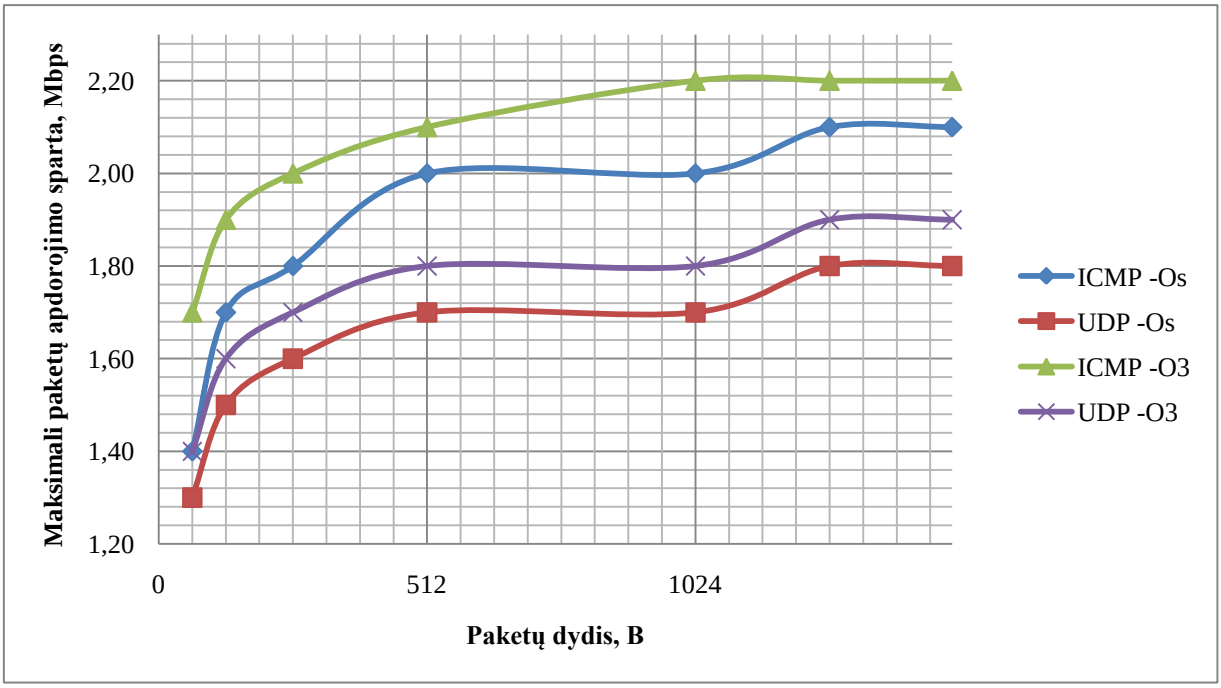
Panaudojus –Os kompiliavimo lygį rezultatai parodyti 4.5 lentelėje.

4.5 lentelė. *ATmega32@16Mhz* paketų apdorojimo rezultatai, kai naudojamas –O3 kompiliavimo lygis

Paketų dydis, B	**ICMP protokolas apdorojimo sparta, Mbps**	**ICMP protokolas paketų vėlinimas, ms**	**UDP protokolas apdorojimo sparta, Mbps**	**UDP protokolas paketų vėlinimas, ms**
64	1,7	0,744	1,4	1
128	1,9	1,092	1,6	1,3
256	2	1,776	1,7	2,1
512	2,1	3,137	1,8	3,6

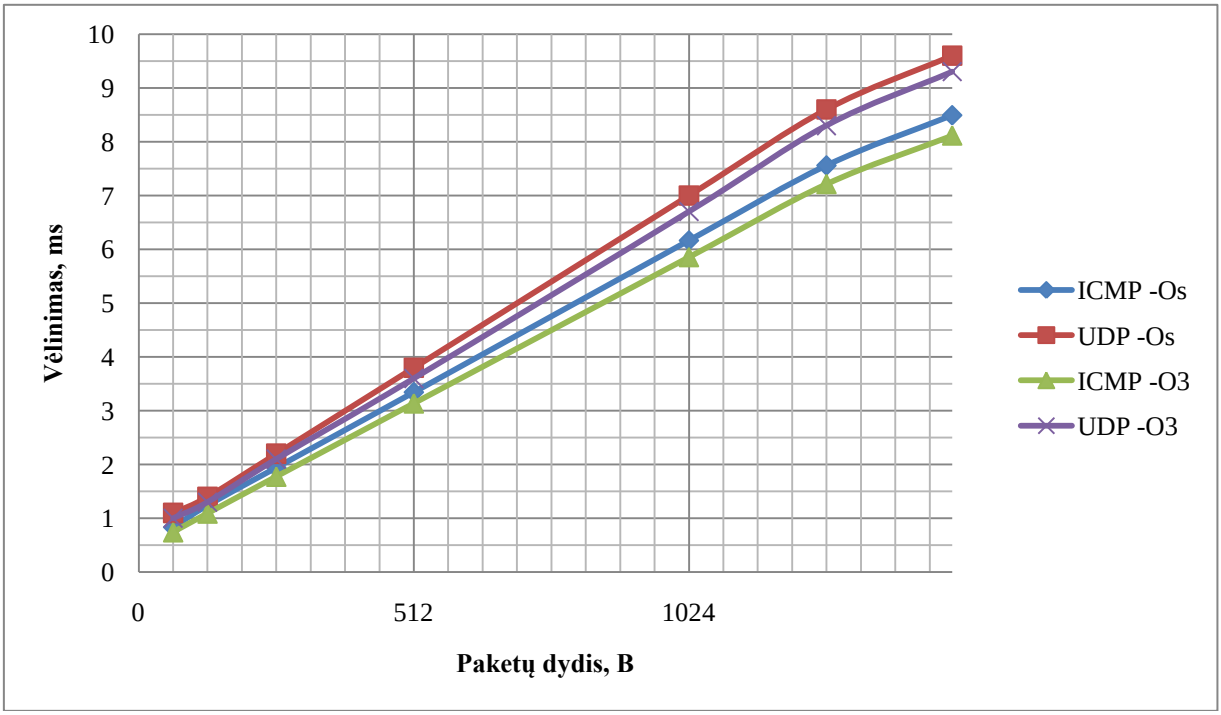
1024	2,2	5,855	1,8	6,7
1280	2,2	7,213	1,9	8,3
1518	2,2	8,114	1,9	9,3

Parinkus kompiliatoriaus kompiliavimo lygį –O3 gauti apie ~5–6 % geresni paketų apdorojimo spartos rezultatai ir apie ~3 % geresni paketų vėlinimo laikai, tačiau programos kodo dydis gaunasi apie 68 % didesnis negu naudojant –Os kompiliavimo lygį. Paketų apdorojimo spartos palyginimas parodytas (4.4 pav.) grafikuose. Grafikuose atvaizduota priklausomybė nuo paketų dydžio.



4.4 pav. Maksimalios paketų apdorojimo spartos priklausomybė nuo paketų dydžio

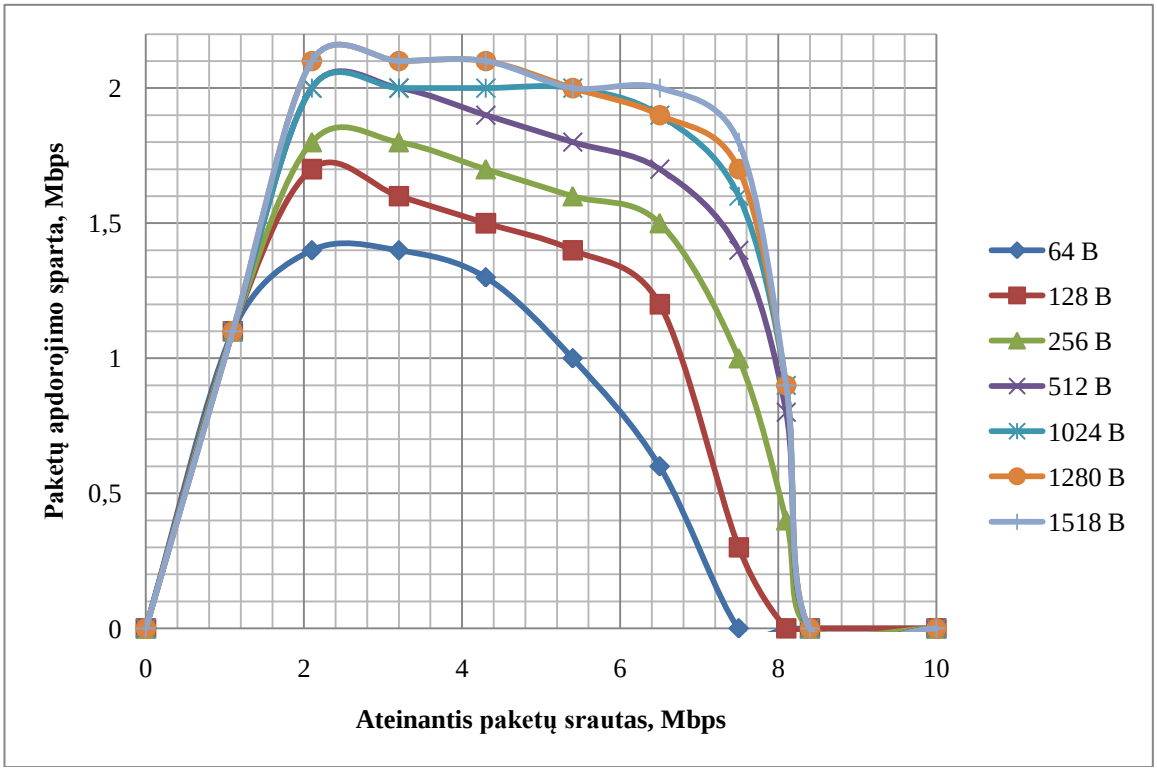
Paketų vėlinimo palyginimas parodytas (4.5 pav.) grafikuose. Grafikuose atvaizduota priklausomybė nuo paketų dydžio.



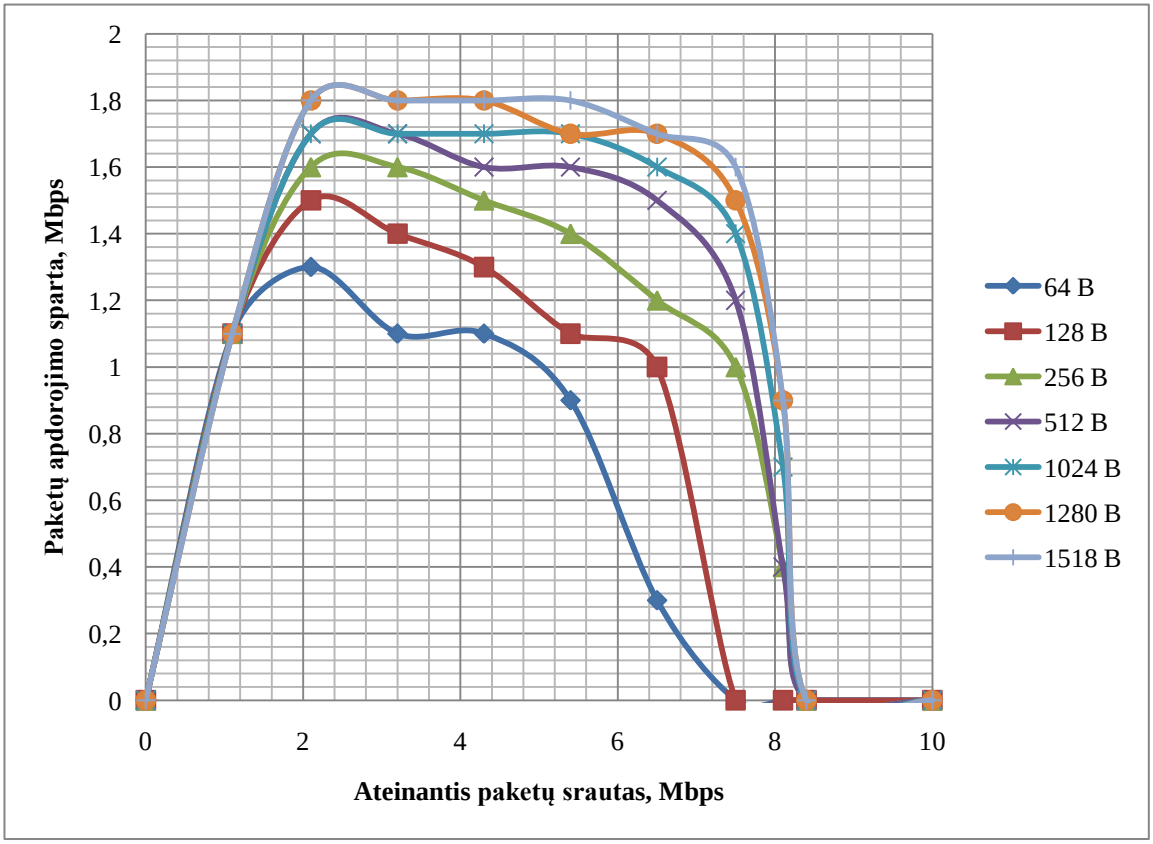
4.5 pav. Paketų vėlinimo priklausomybe nuo paketų dydžio

Paketų apdorojimo sparta priklauso nuo paketo dydžio. Didinant paketo dydį didėja naudingų duomenų dydis, tuo tarpu paketo antgalvio dydis nekinta. Paketo antgalvio apdorojimas užtrunka ilgiau negu duomenų apdorojimas, dėl to mažesnio dydžio paketai yra apdorojami ilgiau negu didesnio dydžio paketai esant tokio pat dydžio antgalviui. Panaudojus –Os kompiliatoriaus kompiliavimo lygį ICMP protokolo paketų apdorojimo sparta kinta nuo 1,4–2,2 Mbps, UDP protokolo 1,3–1,8 Mbps. Esant –O3 kompiliatoriaus kompiliavimo lygiui ICMP protokolo paketų apdorojimo sparta kinta 1,7–2,2 Mbps, UDP protokolo 1,4–1,9 Mbps. Paketų vėlinimas atitinkamai: ICMP protokolui 0,836–8,494 ms (kompiliavimo lygis –Os) ir 0,744–8,114 ms (kompiliavimo lygis –O3), UDP protokolui 1,1–9,6 ms (kompiliavimo lygis –Os) ir 1,0–9,3 ms (kompiliavimo lygis –O3).

Nustatyta mikrovaldiklio paketų apdorojimo spartos priklausomybė nuo ateinančių į mikrovaldiklį paketų srauto dydžio. Didinant ateinančių paketų srauto dydį buvo matuojama mikrovaldiklio paketų apdorojimo sparta. Ateinantis srautas buvo didinamas nuo 1 iki 10 Mbps esant pastoviam paketų dydžiui. Atlikti matavimai skirtingiems paketų dydžiams. ICMP protokolo apdorojimo spartos priklausomybės nuo ateinančio srauto dydžio grafikai parodyti (4.6 pav.). UDP protokolo apdorojimo spartos priklausomybės nuo ateinančio srauto dydžio grafikai parodyti (4.7 pav.).



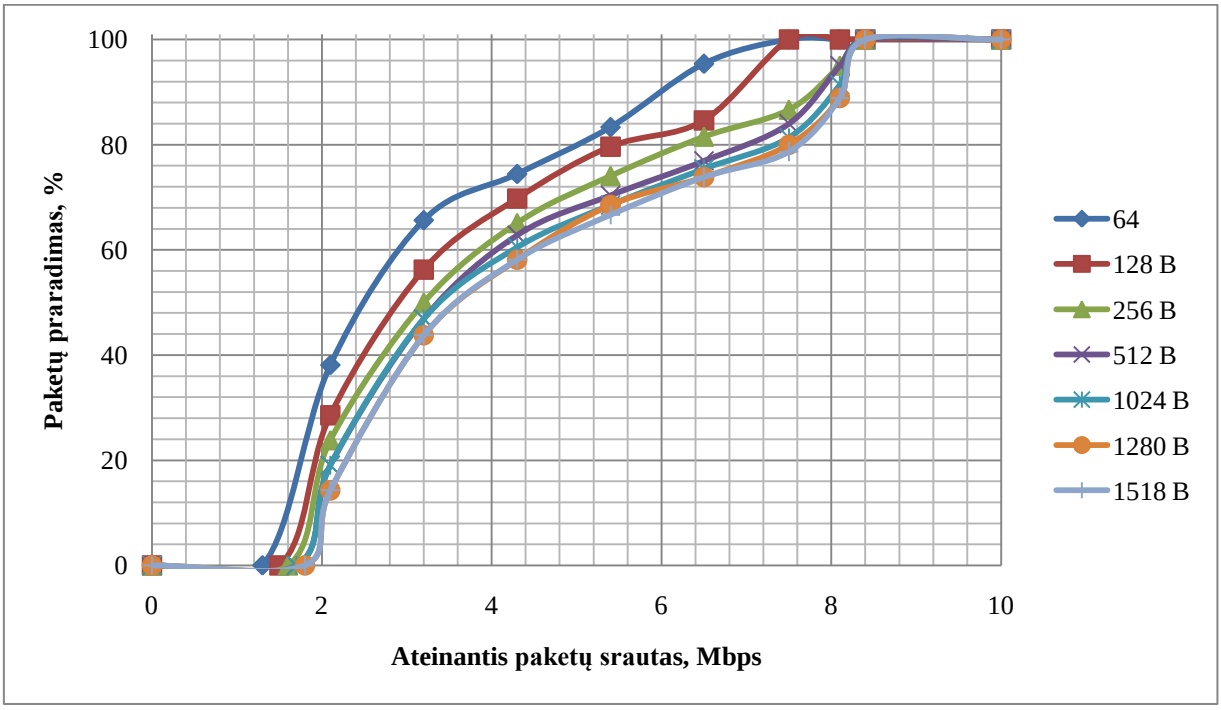
4.6 pav. ICMP protokolo apdorojimo priklausomybė
nuo ateinančio paketų srauto



4.7 pav. UDP protokolo apdorojimo priklausomybė
nuo ateinančio paketų srauto

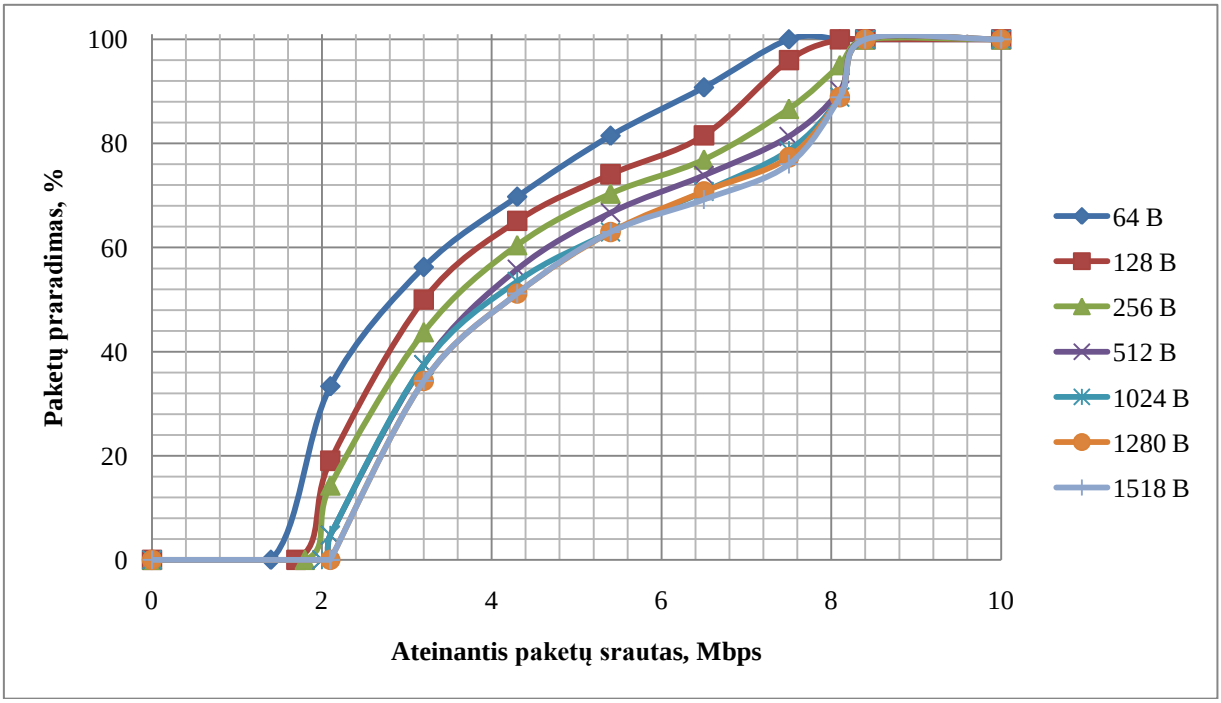
Didinant ateinančių paketų srautą, nepriklausomai nuo pasirinkto protokolo, mikrovaldiklio apdorojimo sparta didėja tiesiškai kol pasiekia maksimalią reikšmę, toliau didinant ateinantį paketų srautą paketai yra atmetami. Didinant ateinančių paketų srautą iki 8 Mbps, mikrovaldiklis visiškai nebespėja apdoroti paketų ir nustoja korektiškai dirbti. 8 Mbps riba yra nustatyta kaip atsparumo DoS atakoms riba.

Iš gautų (4.6 pav.) ir (4.7 pav.) priklausomybių ir maksimalios *Ethernet* sąsajos greičio 10 Mbps, buvo sudaryta paketų praradimo tikimybė. ICMP protokolo paketų praradimo tikimybė parodyta (4.8 pav.). UDP protokolo paketų praradimo tikimybė parodyta (4.9 pav.).



4.8 pav. Skirtingo dydžio ICMP paketų praradimo tikimybė nuo ateinančio paketų srauto

Didinant ateinančių paketų srautą, kol sistemos panaudojimo faktorius < 1 paketų praradimas lygus nuliui. Kai sistemos panaudojimo faktorius > 1 didinant ateinančių paketų srautą, paketų praradimas kintą eksponentinių dėsniu. Pasiekus ateinančiam paketų srautui ~ 8 Mbps greitį paketų praradimo tikimybė išauga iki 100 %. Tokia paketų praradimo charakteristika nepriklauso nuo pasirinkto protokolo, tokio pat pobūdžio charakteristika gaunama ICMP ir UDP protokolams.

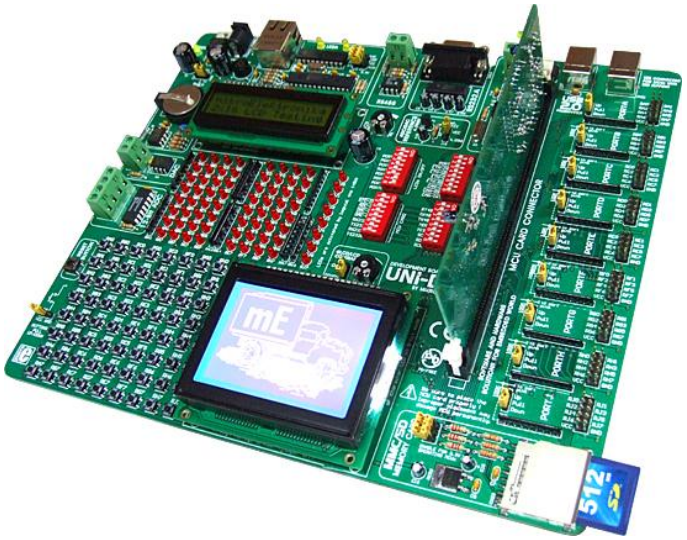


4.9 pav. Skirtingo dydžio UDP paketų praradimo tikimybė nuo ateinančio paketų srauto

Šių paketų praradimo tikimybės charakteristikos sutampa su teoriškai apskaičiuotomis charakteristikomis, kai sistema buvo modeliuojama kaip masinio aptarnavimo sistemą su apribota eile. Gauti praktiniai ir teoriniai rezultatai sutampa.

Antras tyrimas

Panaudojus tą pačia aparatinę bazę, tą patį mikrovaldiklį ištirta paketų apdorojimo spartos priklausomybė nuo panaudoto programos algoritmo. Palygintos dvi programos parašytas panaudojus skirtingus kompiliatorius *AVR-GCC* ir *mikroC*. Kaip aparatinė bazė buvo parinktas tyrimo maketas *mikroElektronika UNI-DS3*, pasirinktas mikrovaldiklis *ATmega128@10MHz* (4.11 pav.) [27]. Maketas *mikroElektronika UNI-DS3* parodytas (4.10 pav.).



4.10 pav. *mikroElektronika UNI-DS3* tyrimo maketas

4.11 pav. *ATmega128* mikrovaldiklis

Žemiau 4.6 lentelėje pateikti tyrimo rezultatai, kai panaudota programa parašyta C# programavimo kalbą, naudojamas adaptuotas algoritmas ir *AVR-GCC* kompiliatorius.

4.6 lentelė. *ATmega128@10Mhz* paketų apdorojimo rezultatai, kompiliatorius *AVR-GCC*

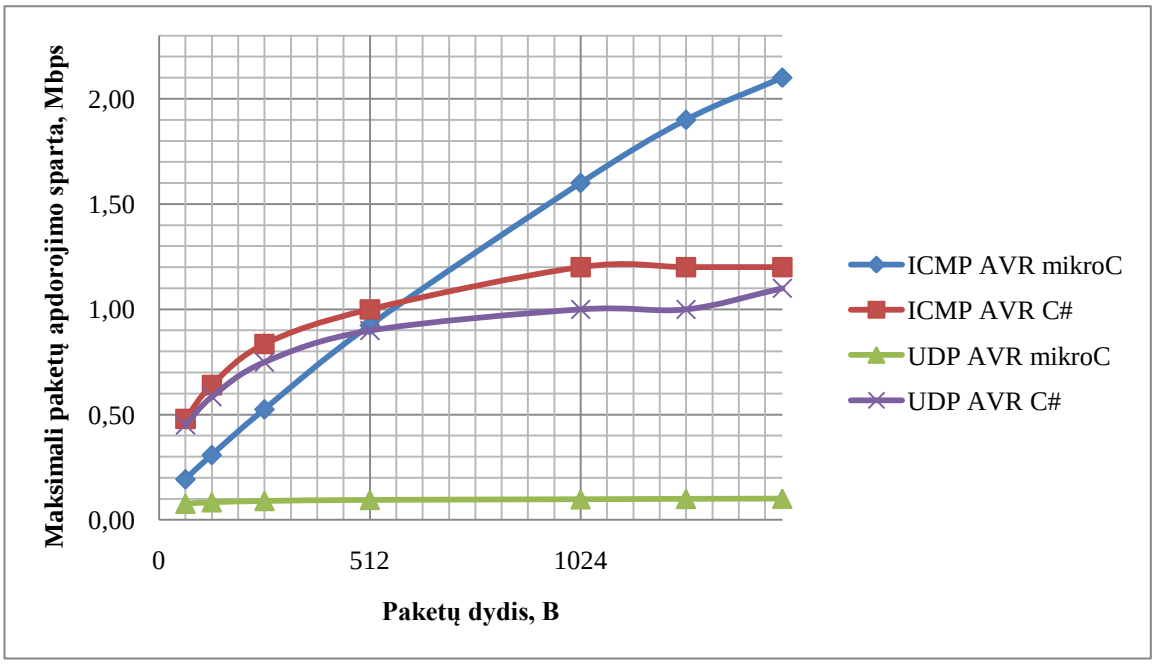
Paketų dydis, B	**ICMP protokolas apdorojimo sparta, Mbps**	**ICMP protokolas paketų vėlinimas, ms**	**UDP protokolas apdorojimo sparta, Mbps**	**UDP protokolas paketų vėlinimas, ms**
64	0,48	2,071	0,451	2,3
128	0,64	2,551	0,584	3
256	0,835	3,514	0,749	4
512	1	5,438	0,899	6,2
1024	1,2	9,289	1	13
1280	1,2	11,206	1	13,9
1518	1,2	12,222	1,1	13,9

Žemiau 4.7 lentelėje pateikti tyrimo rezultatai, kai panaudota programa parašyta *mikroC* programavimo kalbą, naudojama pilna TCP/IP steko biblioteka ir naudojamas *mikroC PRO for AVR 2009* kompiliatorius.

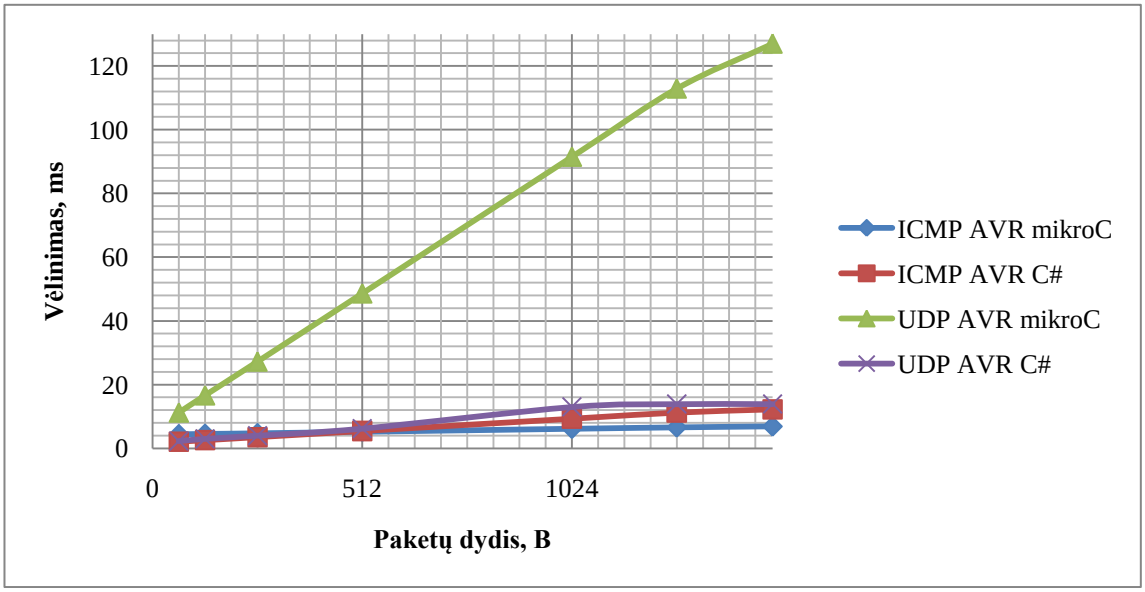
4.7 lentelė *ATmega128@10Mhz* paketų apdorojimo rezultatai, kompiliatorius *mikroC PRO for AVR 2009*

Paketų dydis, B	ICMP protokolas apdorojimo sparta, Mbps	ICMP protokolas paketų vėlinimas, ms	UDP protokolas apdorojimo sparta, Mbps	UDP protokolas paketų vėlinimas, ms
64	0,19	4,414	0,077	11,2
128	0,31	4,551	0,084	16,6
256	0,53	4,732	0,09	27,2
512	0,92	5,207	0,095	48,6
1024	1,60	6,136	0,098	91,5
1280	1,90	6,571	0,1	112,9
1518	2,10	6,886	0,101	127

Paketų apdorojimo spartos ir vėlinimo palyginimas parodytas (4.12 pav.) ir (4.13 pav.) grafikuose. Grafikuose atvaizduota priklausomybė nuo paketų dydžio.



4.12 pav. Maksimalios paketų apdorojimo spartos priklausomybė nuo paketų dydžio



4.13 pav. Paketų vėlinimo priklausomybė nuo paketų dydžio

Iš tyrimo rezultatų ir gautų priklausomybių (4.12 pav.) galima padaryti tokias išvadas. Panaudojus skirtingas programavimo kalbas, programų algoritmus ir skirtingus kompiliatorius, gautos skirtingos paketų apdorojimo charakteristikos. Naudojant *C#* kalbą parašytą programą gautos kvadratinės šaknies pobūdžio priklausomybės panašios abiem naudojamiems protokolams ICMP ir UDP. Santykinai ICMP protokolo apdorojimo sparta yra didesne negu UDP protokolo, priklausomybių kreivių nuo paketų dydžio pabudįs yra panašus. Naudojant *mikroC* kalbą parašytą programą gauti skirtingi rezultatai skirtingiems protokolams. ICMP protokolo apdorojimo priklausomybė nuo paketo dydžio yra tiesinio pobūdžio. UDP protokolo apdorojimo priklausomybė nuo paketo dydžio beveik nekinta, bet paketų vėlinimas labai priklauso nuo paketų dydžio ir kinta 11,2—127 ms. ICMP paketų apdorojimo sparta didžiausia kai paketo dydis yra 1518 B ir kai buvo panaudota *mikroC* programavimo kalba, apdorojimo sparta siekia 2,1 Mbps. UDP paketų apdorojimo sparta didžiausia kai paketo dydis yra 1518 B ir kai buvo panaudota *C#* programavimo kalbą, apdorojimo sparta siekia 1,1 Mbps.

Trečias tyrimas

Atliktas dviejų skirtingų mikrovaldiklių tyrimas ATmega128@10MHz (4.11 pav.) ir PIC16F877A@10MHz (4.14 pav.) panaudojus vienodą programos algoritmą, programavimo kalbą ir kompiliatorių. Panaudota ta pati aparatinė bazė, maketas *mikroElektronika UNI-DS3* parodytas (4.10 pav.) [27].

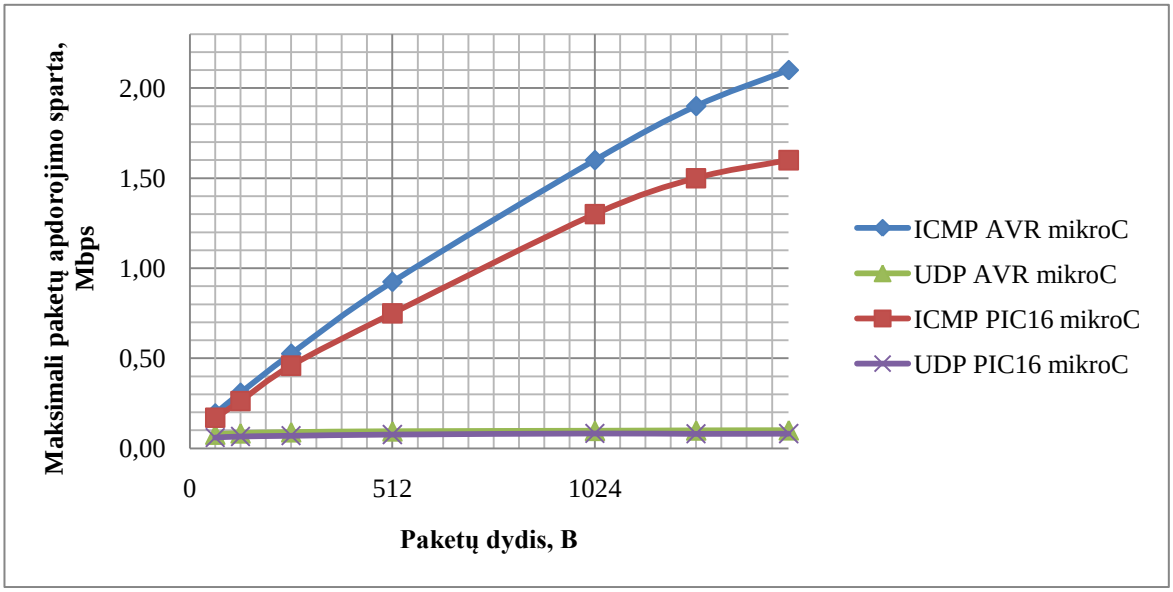
4.14 pav. *PIC16F877A* mikrovaldiklis

ATmega128@10MHz tyrimo rezultatai pateikti ankščiau 4.7 lentelėje. *PIC16F877A@10MHz* tyrimo rezultatai pateikti 4.8 lentelėje. Abu mikrovaldikliai yra aštuonių skilčių, naudoja 10 MHz taktinio dažnio kvarcinį rezonatorių.

4.8 lentelė. *PIC16F877A@10MHz* paketų apdorojimo rezultatai, *mikroC* programavimo kalba

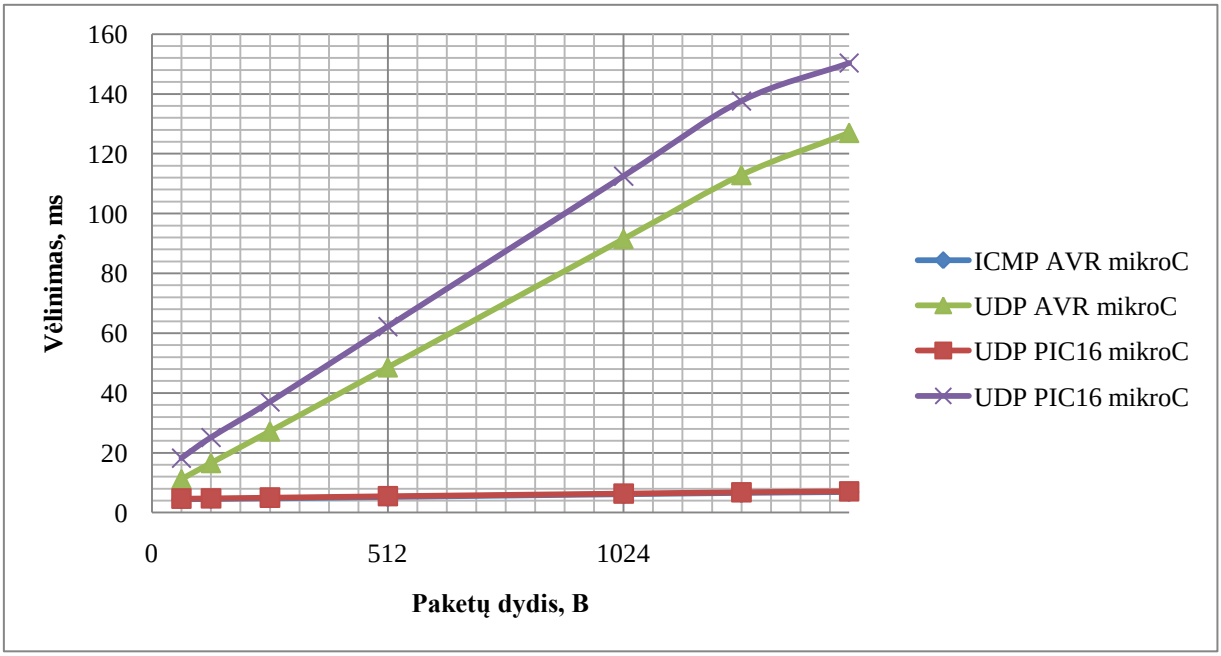
Paketų dydis, B	**ICMP protokolas apdorojimo sparta, Mbps**	**ICMP protokolas paketų vėlinimas, ms**	**UDP protokolas apdorojimo sparta, Mbps**	**UDP protokolas paketų vėlinimas, ms**
64	0,17	4,695	0,06	18,2
128	0,263	4,794	0,066	25,1
256	0,459	5,015	0,07	37,0
512	0,749	5,494	0,077	62,2
1024	1,3	6,353	0,083	112,5
1280	1,5	6,794	0,081	137,6
1518	1,6	7,092	0,082	150,3

Paketų apdorojimo spartos ir vėlinimo palyginimas parodytas (4.15 pav.) ir (4.16 pav.) grafikuose. Grafikuose atvaizduota priklausomybė nuo paketų dydžio.



4.15 pav. Maksimalios paketų apdorojimo spartos priklausomybė nuo paketų dydžio

Iš gautų paketų apdorojimo priklausomybių (4.15 pav.) matome, kad ICMP protokolo paketų apdorojimo priklausomybės kinta pagal tiesinį dėsnį. Mikrovaldiklis *ATmega128* šį protokolą apdoroja greičiau, esant tokiam pat paketų dydžiui 1518 B paketų apdorojimo sparta siekia 2,1 Mbps, tuo tarpu *PIC16F877A* mikrovaldikliui siekia iki 1,6 Mbps. Tokie rezultatai gauti dėl to, kad AVR šeimos mikrovaldikliai vykdo programos komandas per vieną taktinį impulsą, tuomet PIC šeimos mikrovaldikliai vykdo komandas per vieną mašininius ciklą, kuris užtrunka keturis taktinius impulsus, dėl to PIC mikrovaldiklių našumas prie 10 MHz rezonatoriaus yra mažesnis palyginus su AVR šeimos mikrovaldikliais. UDP protokolo apdorojimo sparta nepriklauso nuo paketų dydžio ir yra pastovi, taip pat nepriklauso nuo mikrovaldiklio tipo. Tai gali būti *mikroElektronika* TCP/IP bibliotekos ypatumai.



4.16 pav. Paketų vėlinimo priklausomybė nuo paketų dydžio

Iš paketų vėlinimo charakteristikos (4.16 pav.) matome, kad ICMP protokolo paketų vėlinimas priklauso nuo paketų dydžio. UDP protokolo paketų vėlinimas didėjant paketų dydžiui sparčiai didėja. *ATmega128* mikrovldikliui pasiekia 127 ms, *PIC16F877A* mikrovldikliui pasiekia 150,3 ms.

Ketvirtas tyrimas

PIC16F877A@10MHz mikrovaldiklio paketų apdorojimo spartos tyrimas panaudojus, vienodą programos algoritmą parašytą skirtingomis programavimo kalbomis: *mikroC*, *mikroPascal* ir *mikroBasic*. Mikrovaldiklio paketų apdorojimo spartos rezultatai panaudojus *mikroC* programavimo kalbą pateikti 4.8. lentelėje, panaudojus *mikroPascal* pateikti 4.9 lentelėje ir panaudojus *mikroBasic* pateikti 4.10. lentelėje.

4.9 lentelė. *PIC16F877A@10MHz* paketų apdorojimo rezultatai, *mikroPascal* programavimo kalba

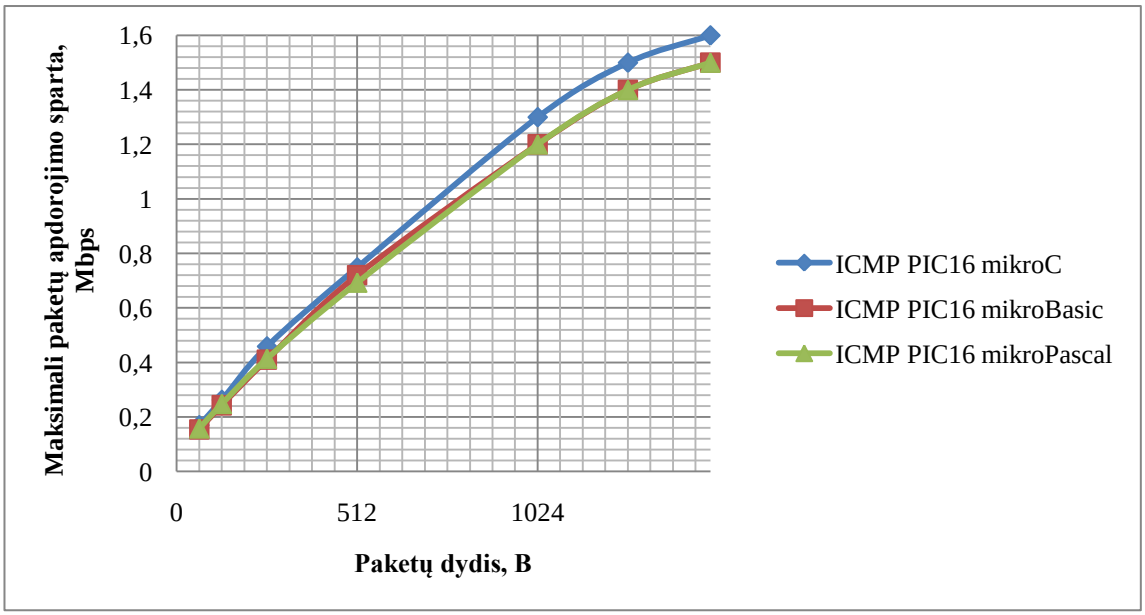
Paketų dydis, B	**ICMP protokolas apdorojimo sparta, Mbps**	**ICMP protokolas paketų vėlinimas, ms**	**UDP protokolas apdorojimo sparta, Mbps**	**UDP protokolas paketų vėlinimas, ms**
64	0,158	5,173	0,065	13,3
128	0,248	5,3	0,071	19,7
256	0,414	5,485	0,074	32,6
512	0,693	5,933	0,076	58,5

1024	1,2	6,864	0,081	109,6
1280	1,4	7,261	0,084	135,2
1518	1,5	7,546	0,083	148,7

4.10 lentelė. *PIC16F877A@10MHz* paketų apdorojimo rezultatai, *mikroBasic* programavimo kalba

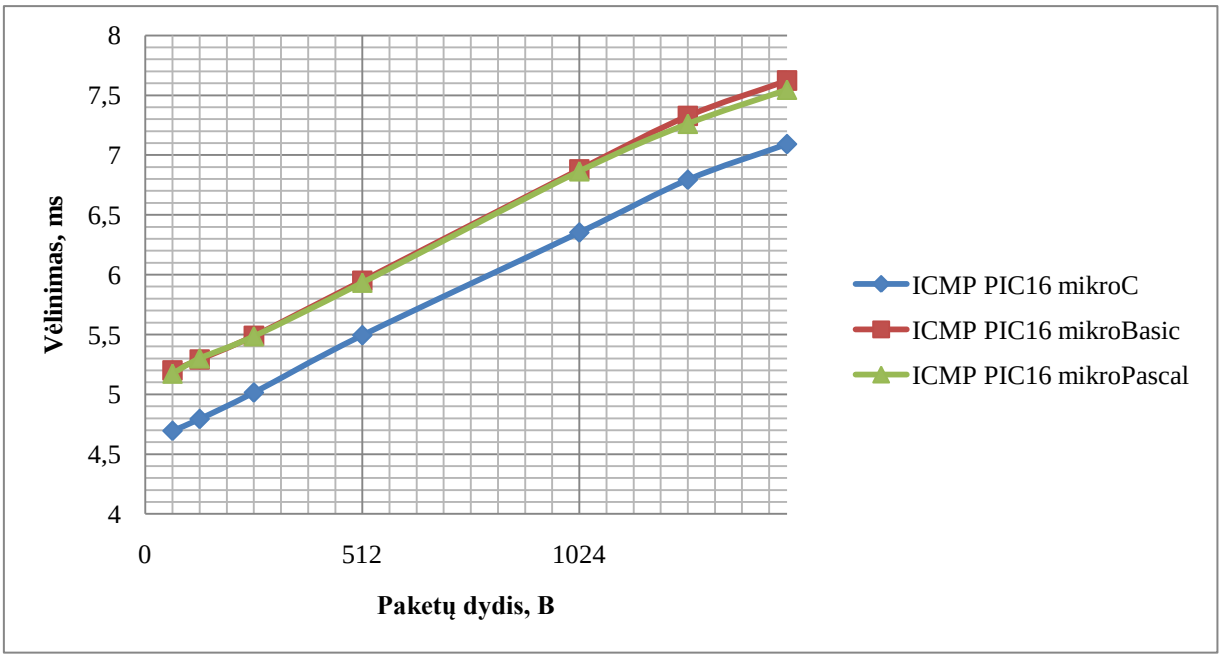
Paketų dydis, B	**ICMP protokolas apdorojimo sparta, Mbps**	**ICMP protokolas paketų vėlinimas, ms**	**UDP protokolas apdorojimo sparta, Mbps**	**UDP protokolas paketų vėlinimas, ms**
64	0,154	5,2	0,064	13,3
128	0,243	5,291	0,07	19,8
256	0,41	5,489	0,075	33
512	0,72	5,95	0,077	58,2
1024	1,2	6,88	0,081	109,5
1280	1,4	7,327	0,079	135,2
1518	1,5	7,623	0,077	154,2

Paketų apdorojimo spartos palyginimas ICMP protokolui priklausomai nuo panaudotos programavimo kalbos parodytas (4.17 pav.) grafike. Grafike atvaizduota priklausomybė nuo paketų dydžio.



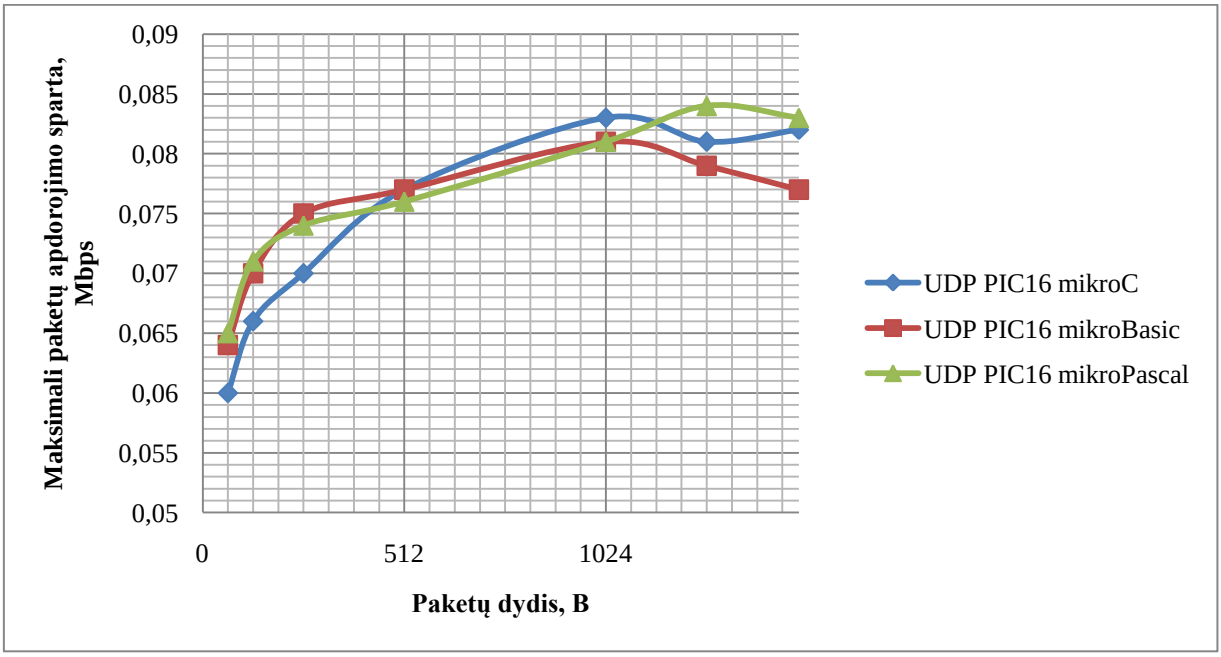
4.17 pav. Maksimalios paketų apdorojimo spartos priklausomybė nuo paketų dydžio

Iš gautų priklausomybių matome, kad algoritmas parašytas *mikroBasic* ir *mikroPascal* kalbomis ir panaudotas ICMP paketų apdorojimui duoda tokius pat rezultatus. Panaudojus *mikroC* programavimo kalbą parašytą algoritmą gauti apie 7 % geresni rezultatai. Galime padaryti išvadą, kad programavimo kalbos pasirinkimas turi įtakos algoritmo veikimo spartai. Taip pat *mikroC* kalba parašyto algoritmo veikimo pranašumas matomas ir paketų vėlinimo priklausomybių grafike (4.18 pav.).



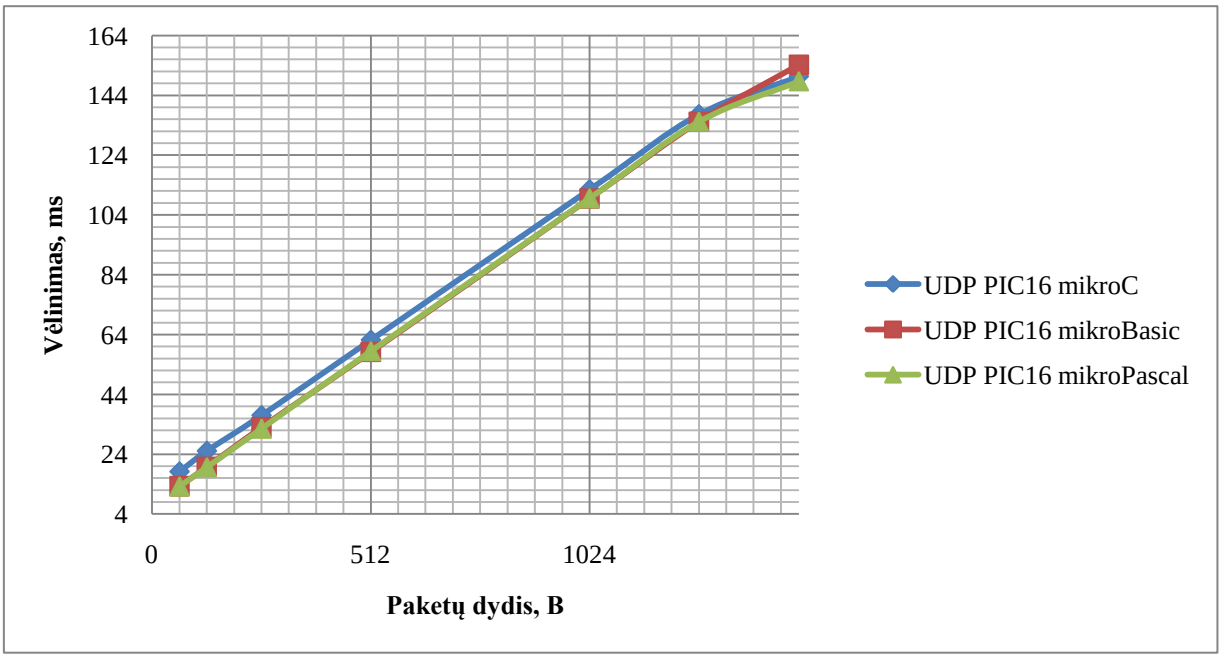
4.18 pav. Paketų vėlinimo priklausomybė nuo paketų dydžio

Paketų apdorojimo spartos palyginimas UDP protokolui priklausomai nuo panaudotos programavimo kalbos parodytas (4.19 pav.) grafike. Grafike atvaizduota priklausomybė nuo paketų dydžio.



4.19 pav. Maksimalios paketų apdorojimo spartos priklausomybė nuo paketų dydžio

Parašytas skirtingomis programavimo kalbomis tas pats algoritmas ir panaudotas UDP protokolui apdoroti neduoda jokio pranašumo ir ne priklauso nuo pasirinktos programavimo kalbos. Tokį pat rezultatą gauname ir UDP protokolo paketų vėlinimo priklausomybėje nuo paketų dydžio (4.20 pav.).



4.20 pav. Paketų vėlinimo priklausomybė nuo paketų dydžio

Tokie UDP protokolo rezultatai gauti dėl panaudotos *mikroElektronika* TCP/IP bibliotekos ypatumų.

Penktas tyrimas

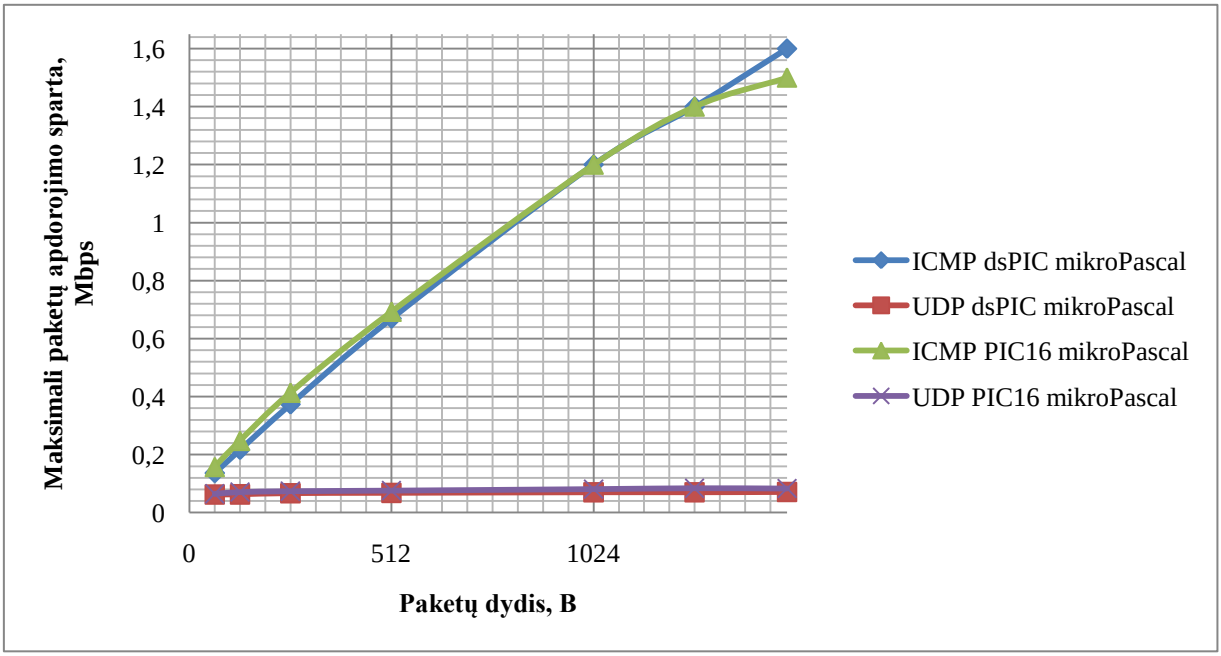
Šiame tyrime atliktas aštuonių skilčių mikrovaldiklių paketų apdorojimo spartos rezultatų palyginimas su 16 skilčių mikrovaldikliu *dsPIC30F6014A@10MHz* (4.21 pav.) [23]. Paketų apdorojimo spartos rezultatai pateikti žemiau 4.11 lentelėje.

4.21 pav. *PIC16F877A* mikrovaldiklis

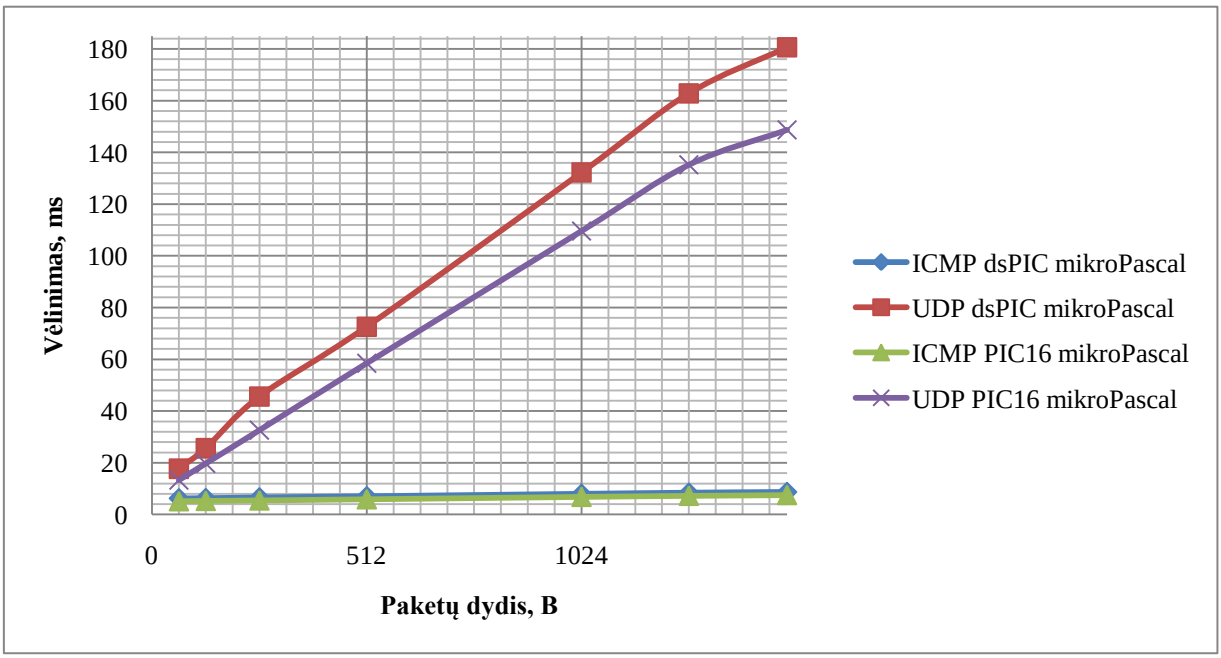
4.11 lentelė. *dsPIC30F6014A@10MHz* paketų apdorojimo rezultatai

Paketų dydis, B	ICMP protokolas apdorojimo sparta, Mbps	ICMP protokolas paketų vėlinimas, ms	UDP protokolas apdorojimo sparta, Mbps	UDP protokolas paketų vėlinimas, ms
64	0,136	6,226	0,062	17,7
128	0,216	6,306	0,063	25,6
256	0,373	6,572	0,067	45,6
512	0,669	7,018	0,068	72,6
1024	1,2	7,964	0,07	132,2
1280	1,4	8,407	0,07	162,8
1518	1,6	8,684	0,071	180,6

Gautus rezultatus palyginsime su *PIC16F877A@10MHz* 1.9. lentelėje pateiktais rezultatais. *PIC16* ir *dsPIC30* yra tos pačios šeimos mikrovaldikliai skiriasi tiktai skiltiškumas, programos parašytos naudojant tą patį algoritmą ir ta pačią programavimo kalbą *mikroPascal*. Paketų apdorojimo spartos ir paketų vėlinimo priklausomybės parodytos (4.22 pav.) ir (4.23. pav.).



4.22 pav. Maksimalios paketų apdorojimo spartos priklausomybė nuo paketų dydžio



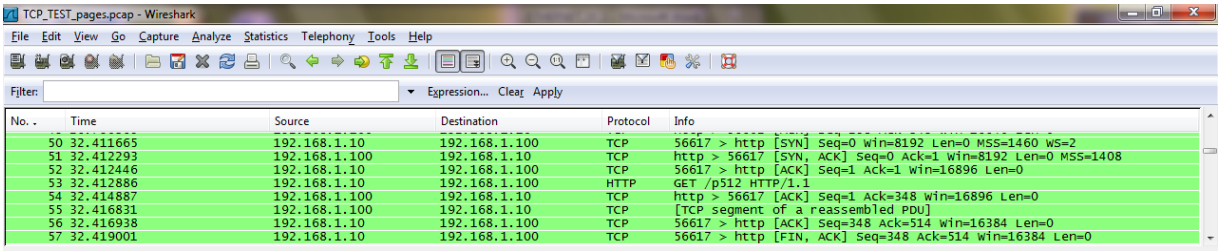
4.23 pav. Paketų vėlinimo priklausomybė nuo paketų dydžio

Iš gautų paketų apdorojimo spartos rezultatu matome, kad ICMP ir UDP protokolų paketų apdorojimo sparta beveik nepriklauso nuo pasirinkto mikrovaldiklio skiltiškumo, tai gali būti dėl neadaptuotos *Ethernet* valdiklio bibliotekos 16 skilčių mikrovaldikliui. Dėl to 16 skilčių mikrovaldiklis neturi tokio ženklaus spartos pranašumo. Iš paketų vėlinimo priklausomybių grafikų matome, kad ICMP protokolo paketų vėlinimo laikai beveik sutampa,

tačiau UDP protokolo paketų vėlinimas panaudojus *dsPIC30F6014A* mikrovaldiklį žymiai didesnis už vėlinimą panaudojus *PIC16F877A* aštuonių skilčių mikrovaldiklį.

Šeštas tyrimas

Atliktas TCP ir HTTP protokolų kompleksinis tyrimas panaudojus *ATmega32@16Mhz* mikrovaldiklį. Tyrimas apima transporto ir taikomųjų programų lygmenis. Tai yra TCP transporto lygmens protokolas ir HTTP taikomųjų programų lygmens protokolas. Šis tyrimas yra kompleksinis apimantis du lygmenis, šio tyrimo rezultatas atvaizduos HTML puslapio užkrovimo laiką priklausomai nuo puslapio dydžio. Mikrovaldiklio programoje sudaromi skiurtingo dydžio HTML puslapiai (64, 128, 256, 512, 1024, 1280, 1518 B) kiekvienas iš puslapių gali būti iškviestas panaudojus individualią nuorodą naršyklėje pvz.: *http://192.168.20.60/p64. Wireshark* paketų analizatoriumi įrašomą visa sesijos užmezgimo istoriją. Iš istorijos nustatomas puslapio užkrovimo laikas nuo pirmosios SYN užklausos iki sesijos pabaigos FIN/ACK. TCP ir HTTP protokolų vienos sesijos fragmentas parodytas (4.24 pav.) [1, 11].



4.24 pav. TCP ir HTTP sesija

Sesija užmezgama siunčiant SYN paketą (4.24 pav. 50), gaunamas SYN/ACK atsakymas (4.24 pav. 51), siunčiamas ACK atsakymo patvirtinimas (4.24 pav. 52) ir sesija yra užmegzta. Siunčiama puslapio užklausa (HTTP *GET /p512*) (4.24 pav. 53), ACK užklausos patvirtinimas (4.24 pav. 54) ir puslapio duomenis (4.24 pav. 55). Kompiuteris gavus puslapio duomenis tvirtina patvirtinimą ACK (4.24 pav. 56) ir baigia sesiją siunčiant FIN/ACK (4.24 pav. 57). Visas fragmentas parodo TCP sesijos užmezgimą, duomenų perdavimą ir sesijos pabaigą.

Sesijos užmezgimo rezultatai parodyti 4.12 lentelėje.

4.12 lentelė. TCP ir HTTP kompleksinio tyrimo rezultatai

Puslapio dydis, B	**Puslapio užkrovimo laikas, ms**
64	6,198
128	11,592
256	12,347
512	14,950
1024	14,716
1280	14,725
1518	14,647

Iš gautų rezultatu matome, kad puslapio užkrovimo laikas priklauso nuo puslapio dydžio. Kinta 6,198–14,950 ms puslapio dydžiui kintant nuo 64 B iki 1518 B, taip pat matome kad puslapio užkrovimo laikas nebekinta esant 512 B – 1518 B puslapių dydžiui ir yra lygus ~14,7 ms.

Iš atliktų tyrimų parinkus skirtingus tyrimo kriterijus buvo nustatytos aštuonių skilčių mikrovaldiklių paketų apdorojimo spartos ir jų priklausomybės nuo paketų dydžio. Kaip matome iš rezultatų paketų apdorojimo sparta priklauso nuo mikrovaldiklio šeimos, programos algoritmo, panaudotos programavimo kalbos, kompiliatoriaus nustatymų.

4.4. Pasiūlymai

Projektuojant *Ethernet* tinklo įrenginį kurio pagrindą sudaro aštuonių skilčių mikrovaldiklis, iš ištirtų mikrovaldiklių rekomenduojama naudoti AVR šeimos mikrovaldiklį, nes esant tam pačiam taktiniam kvarcinio rezonatoriaus dažniui mikrovaldiklio sparta yra lygi taktiniam dažniui pvz.: 10 MHz taktiniam dažniui mikrovaldiklio veikimo sparta lygi 10 MIPS.

Programavimui rekomenduojama nauduoti paruoštomis TCP/IP steko bibliotekomis. Rekomenduojama naudoti *C#* programavimo kalba, dėl programos kodo pavyzdžių ir literatūros gausos, bei ir atsižvelgus į tyrimo rezultatus dėl programų veikimo spartos parašytų *C#* programavimo kalba. Naudojant *C#* programavimo kalba galima naudotis nemokamais kompiliatoriais pvz.: *AVR-GCC* kompiliatorius su didelių bibliotekų pasirinkimų.

Parenkant įrenginio duomenų perdavimo tinkle paketų dydį nepriklausomai nuo protokolo, rekomenduojama naudoti didelius paketus 1000–1500 B eilės tam, kad pasiektume didelį paketų apdorojimo greitį. Ten kur mažiau reikalingas paketų apdorojimo greitis bet svarbus paketų vėlinimas rekomenduojama naudoti mažus paketus 64–256 B eilės.

Apibendrinus gautus tyrimų rezultatus išaiškėjo tokio tipo įrenginių apribojimas naudojant realiuose tinkluose. Įrenginys gali būti naudojamas tokiame tinkle, kur tinklo bendras srautas neviršija 8 Mbps srauto. Šis panaudojimo apribojimas dabartiniuose tinkluose reikštų neįmanomą šio įrenginio realizaciją. Šį apribojimą galima apeiti panaudojus naujų tinklų VLAN technologiją jungiant įrenginius į atskirus potinklius, naudojant ateinančio į įrenginio prievadą srauto ribotuvus. Šie reikalavimai yra nesunku realizuojami su šiuo metu tinkle naudojamais įrenginiais.

IŠVADOS

Pagrindinis šio darbo tikslas *Ethernet* tinklo įrenginių tyrimo metodikos sudarymas ir priemonių parinkimas. Šiame darbe buvo apžvelgti tinklo įrenginiai, kurių pagrindą sudaro aštuonių skilčių mikrovaldikliai. Suprojektuotas įrenginys tyrimui, kurio pagrindą sudaro *ATmega32* mikrovaldiklis ir *ENC28J60 Ethernet* valdiklis. Adaptuotas TCP/IP stekas aštuonių skilčių AVR mikrovaldikliams. Parašyta programa įrenginiui, suprojektuota įrenginio valdymo grafinė aplinka. Sudarytas įrenginio matematinis modelis. Matematiniam modeliui pritaikyta masinio aptarnavimo teorija. Įrenginys sumodeliuotas kaip masinio aptarnavimo sistema su apribota eile. Gautos teorinės paraiškų aptarnavimo priklausomybės nuo sistemos panaudojimo faktoriaus. Sudarytos saugumo, atsparumo atakoms, pagrindinių įrenginio parametrų tyrimo metodikos ir parinktos tyrimo priemonės.

Darbe išsamiai ištirti aštuonių skilčių mikrovaldiklių *Ethernet* tinklo paketų apdorojimo spartos, vėlinimai ir paketų praradimo priklausomybės. Nustatyta šių parametrų priklausomybė nuo mikrovaldiklio šeimos, programos algoritmo, panaudotos programavimo kalbos, kompiliatoriaus nustatymų.

Iš atliktų tyrimų padarytos šios svarbios išvados:

1. Aštuonių skilčių mikrovaldiklių *Ethernet* paketų apdorojimo sparta priklauso nuo programos kodo kompiliatoriaus nustatymų. Palyginus kompiliatoriaus programos kodo optimizavimo lygius, didžiausias apdorojimo spartos skirtumas gautas palyginus du optimizavimo lygius –Os ir –O3. Lygis –Os geriausiai suspaudžia programos kodo dydį dėl vėlinimo ciklų kodo optimizavimo. Lygis –O3 mažiausiai suspaudžia programos kodą. Geriausias rezultatas gautas parinkus –O3 optimizavimo lygį, įrenginio *Ethernet* paketų apdorojimo sparta padidėjo ~6 % vėlinimas sumažėjo ~3 % palyginus su –Os optimizavimo lygių, tačiau programos kodo apimtis padidėjo 68 %.

2. Nustatytos aštuonių skilčių mikrovaldiklių *Ethernet* paketų apdorojimo spartos ir paketų praradimo priklausomybės nuo ateinančio paketų srauto dydžio. Ateinantį paketų srautą didinant nuo 1 iki 10 Mbps esant pastoviam paketų dydžiui kol sistemos panaudojimo faktorius kinta nuo 0 iki 1, didinant paketų srautą mikrovaldiklis jį apdoroja tiesinių dėsniu, paketų praradimas 0 %. Viršijus sistemos panaudojimo faktoriui >1, sistema apdoroja ateinantį srautą pastoviu greičiu. Toliau didinant sistemos panaudojimo faktorių gaunami paketų praradimai. Paketų praradimo priklausomybė nuo sistemos panaudojimo faktoriaus kinta eksponentiniu dėsniu.

3. Didinant ateinančių paketų srautą iki ~8 Mbps paketų praradimo tikimybė išauga iki 100 %, gaunama sistemos atsisakymo aptarnauti būsena. 8 Mbps riba priimta už įrenginio atsparumo atakoms ribą.

4. Palyginti įrenginio modeliuoto kaip masinio aptarnavimo sistema su apribota eile rezultatai su eksperimentiškai gautais rezultatais. Paketų praradimo tikimybė nuo sistemos panaudojimo faktoriaus gauta tokia pat.

5. *Ethernet* paketų apdorojimo sparta priklauso nuo pasirinkto programos algoritmo. Palyginti du programos algoritmai: savarankiškai adaptuotas *AVR-GCC* bibliotekų rinkinys mikrovaldikliams ir *mikroElektronika* siūloma TCP/IP steko bibliotekos panaudojimo demonstracinė programos versija. Paketų apdorojimo priklausomybių kreivės panaudojant skirtingus algoritmus gautos skirtingos, jų kitimo dėsniai yra skirtingi. Panaudojus adaptuotą *AVR-GCC* biblioteką gauta paketų apdorojimo priklausomybė nuo paketų dydžio yra kvadratinės šaknies funkcija. Panaudojus *mikroElektronika* bibliotekos panaudojimo demonstracine versiją paketų apdorojimo priklausomybė nuo paketų dydžio yra tiesinė

6. Paketų apdorojimo sparta priklauso nuo panaudoto mikrovaldiklio tipo. Palygintos dvi aštuonių skilčių mikrovaldiklių šeimos AVR ir PIC mikrovaldikliai. Esant tam pačiam mikrovaldiklių taktiniam dažniui AVR šeimos mikrovaldikliai yra greitesni nes vykdo programos komandas per vieną taktinį impulsą, tuomet PIC šeimos mikrovaldikliai vykdo komandas per vieną mašininius ciklą, kuris užtrunka keturis taktinius impulsus, dėl to PIC šeimos mikrovaldiklių našumas prie to pačio taktinio dažnio yra mažesnis palyginus su AVR šeimos mikrovaldikliais.

7. Paketų apdorojimo sparta priklauso nuo panaudotos programavimo kalbos. Buvo ištirtos *mikroC, mikroPascal* ir *mikroBasic* programavimo kalbos. Panaudojus *mikroPascal* ir *mikroBasic* kalbas paketų apdorojimo rezultatai beveik nesiskiria. Panaudojus *mikroC* programavimo kalbą gauti rezultatai esant 1518 B paketų dydžiui yra ~ 6 % geresni, mažėjant paketų dydžiui šis skirtumas mažėja.

Apibendrinant padarytas išvadas galima pasakyti, kad aštuonių skilčių mikrovaldiklius galima naudoti *Ethernet* tinklo įrenginiuose, ten kur bendra tinklo apkrova neviršija 8 Mbps srauto. Maksimali įrenginio tinklo paketų apdorojimo sparta priklauso nuo paketų dydžio ir priklausomai nuo naudojamo protokolo gali siekti iki 2 Mbps. Papildomai atsižvelgus į skirtas rekomendacijas šie įrenginiai gali būti naudojami tinkluose su dideliais duomenų srautais. Tokių įrenginių spartos pakanka norint panaudoti naudotojo grafinę aplinką ir realizuoti įvairių sąsajų konvertavimą į *Ethernet* sąsają.

LITERATŪRA

1. Walter Goralski, The Illustrated Network. Wow TCP/IP Works In A Modern Network. London: 2009. 829p.
2. Donald Gross, Carl M. Harris, Fundamentals of Queueing Theory. Hangover: 1998. 464p.
3. Rimantas Plėštys, R. Kavaliūnas, G. Vilutis, I. Lagzdynytė, V. Liutkauskas, Kompiuterių tinklai. Kaunas: 2008. 152p.
4. Richard H. Barnett, Embedded C Programming And The Atmel AVR, 2006. New York: 560p.
5. Mark G. Sobell, A Practical Guide to Ubuntu Linux. Massachusetts: 2007. 1141p.
6. Sander van Vugt, Beginning the Linux Command Line. New York: 2009. 381p.
7. A. Baškys, Mikrokompiuteriai. Vilnius: „Technika", 2006. 164 p.
8. Ethernet matavimų metodikos standartas RFC 2544. Prieiga per internetą: http://www.ietf.org/rfc/rfc2544.txt [žiūrėta 2010.06.01]
9. AVR Studio 4 kompiliatoriaus aprašymas. Prieiga per internetą: http://www.engr.sjsu.edu/bjfurman/courses/ME106/ME106pdf/UsingAVRStudio4.pdf [žiūrėta 2010.06.01]
10. AVR-GCC library. Prieiga per internetą: http://www.nongnu.org/avr-libc/ [žiūrėta 2010.06.01]
11. *Hping3* įrankis. Prieiga per internetą: http://www.hping.org/ [žiūrėta 2010.06.01]
12. *Nmap* įrankis. Prieiga per internetą: http://nmap.org/ [žiūrėta 2010.06.01]
13. *Ping* įrankis. Prieiga per internetą: http://linux.about.com/od/commands/l/blcmdl8_ping.htm [žiūrėta 2010.06.01]
14. *Wireshark* įrankis. Prieiga per internetą: http://www.wireshark.org/ [žiūrėta 2010.06.01]
15. *DoS* atakos. Prieiga per internetą: http://en.wikipedia.org/wiki/Denial-of-service_attack [žiūrėta 2010.06.01]
16. *SYN flood* ataka. Prieiga per internetą: http://en.wikipedia.org/wiki/SYN_flood
17. *UDP flood* ataka. Prieiga per internetą: http://en.wikipedia.org/wiki/UDP_flood_attack [žiūrėta 2010.06.01]
18. *Fiveco* gamintojo puslapis. Prieiga per internetą: http://www.fiveco.ch [žiūrėta 2010.06.01]

19. *Konerix* gamintojo puslapis. Prieiga per internetą: http://www.korenix.com [žiūrėta 2010.06.01]
20. *Moxa* gamintojo puslapis. Prieiga per internetą: http://www.moxa.com [žiūrėta 2010.06.01]
21. Stand-Alone *Ethernet Controller with SPI™ Interface ENC28J60* gamintojo puslapis. Prieiga per internetą: http://ww1.microchip.com/downloads/en/DeviceDoc/39662c.pdf [žiūrėta 2010.06.01]
22. *Microchip* gamintojo *Ethernet* technologijos apžvalga. Prieiga per internetą: http://ww1.microchip.com/downloads/en/AppNotes/01120a.pdf [žiūrėta 2010.06.01]
23. *Atmel* gamintojo puslapis. *ATmega32* mikrovaldiklio aprašymas. Prieiga per internetą: http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf [žiūrėta 2010.06.01]
24. *Atmel* gamintojo puslapis. *Two-wire Serial EEPROM* aprašymas. Prieiga per internetą: http://www.atmel.com/dyn/resources/prod_documents/doc0180.pdf [žiūrėta 2010.06.01]
25. HTML skriptinio programavimo kalbos žinynas. Prieiga per internetą: http://html.manual.ru/ [žiūrėta 2010.06.01]
26. CSS skriptinio programavimo kalbos žinynas. Prieiga per internetą: http://css.manual.ru/ [žiūrėta 2010.06.01]
27. MikroElektronika UNI-DS3 tyrimų plokšte. Prieiga per internetą: http://www.mikroe.com/eng/products/view/421/uni-ds3-development-system/ [žiūrėta 2010.06.01]